FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Framework for genomic based cancer studies using Machine Learning algorithms

João Loureiro

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Camacho (FEUP)

Co-Supervisor: Luís Teixeira (FEUP)

March 18, 2019

Framework for genomic based cancer studies using Machine Learning algorithms

João Loureiro

Mestrado Integrado em Engenharia Informática e Computação

March 18, 2019

Abstract

The health sciences are a fundamental asset of knowledge based support for proper and efficient delivery of health care to human beings. Nowadays *Bioinformatics* [8] play a very important roles within the Life Sciences as they are responsible for developing methods and applying tools of computation and analysis to extract knowledge from biological data. [16]

In the globalizing world, knowledge and information are seen as keys of power and a new trading good. [52] Information is widely spread and public, the main challenge to extract usable and needed knowledge is in the diversity, volume and complexity of the available data. Particularly in *Bioinformatics* and computational biology we face not only increased volume and a diversity of highly complex, multidimensional and often weakly-structured and noisy data, but also the growing need for integrative analysis and modeling. [26] [38]

Genomics [39] is an interdisciplinary field of biology that focus on the study of genomes structure, function, evolution, mapping, and editing. With the amount of data available about human DNA [5] genomic research, scientists and clinicians are nowadays applying this knowledge to study the role that multiple genetic factors have on complex diseases, such as cancer, diabetes, and cardiovascular disorders.

The goal of this project is to test if a framework that allows the application of Machine Learning to the analysis of genomic data can be an advantage to process and extract knowledge from this data. This framework must be able to deal with data from different types of analysis and experiments, allowing the scientist to process, organize and visualize the data, build Machine Learning models trained using that data, evaluate the results and finally produce a report like document with the entire process.

At the end we expect that our approach can help geneticist and be used to improve the health care system.

Resumo

As ciências da saúde são um recurso fundamental no apoio, baseado em conhecimento, para a entrega adequada e eficiente de cuidados de saúde a seres humanos. Atualmente, a Bioinformática [8] desempenha um papel muito importante dentro das Ciências da Vida, pois são responsáveis pelo desenvolvimento de métodos e pela aplicação de ferramentas de computação e análise para extrair conhecimento de dados biológicos. [16]

Num mundo cada vez mais globalizado, o conhecimento e informação são vistos como objectos de poder e um novo bem comercial. [52] A informação é amplamente difundida e pública, o principal desafio para extrair conhecimento utilizável e necessário é a diversidade, volume e complexidade dos dados disponíveis. Particularmente em Bioinformática e biologia computacional, enfrentamos não só um aumento do volume e uma diversidade de dados altamente complexos, multidimensionais e muitas vezes fracamente estruturados e ruidosos, mas também a necessidade crescente de análise e modelação continua. [26] [38]

Genómica [39] é um campo interdisciplinar da biologia que se concentra no estudo da estrutura, função, evolução, mapeamento e edição dos genomas. Com a quantidade de dados disponíveis sobre a pesquisa genómica de DNA [5] humano, cientistas e clínicos estão a aplicar este conhecimento para estudar o papel que múltiplos fatores genéticos têm em doenças complexas, como cancro, diabetes e doenças cardiovasculares.

O objetivo deste projeto é testar se uma framework que permite a aplicação do Machine Learning à análise de dados genómicos pode ser uma vantagem para processar e extrair conhecimento desses dados. Esta ferramenta deve ser capaz de lidar com dados de diferentes tipos de análise e experiências, permitindo que o cientista processe, organize e visualize os dados, construa modelos de Machine Learning usando esses dados, avalie os resultados e finalmente produza um relatório com o processo inteiro.

No final, esperamos que nossa abordagem possa ajudar os geneticistas e ser usada para melhorar o sistema de saúde.

Acknowledgements

First of all, I want to thank my supervisor Rui Camacho, without whom I would never be able to work with this project. His guidance, ideas, support and readiness were key factors to the delivery of this thesis. My gratitude to my co-supervisor Luís Teixeira for all the feedback and help during this semester.

I want to thank the Macro-to-Nano Human Sensing: Towards Integrated Multimodal Health Monitoring(NanoSTIMA) and Analytics/NORTE-01-0145-FEDER-000016 projects. They are funded by the North Regional Operational Program (NORTE 2020), under the partnership agreement PORTUGAL 2020 and the European Regional Development Fund - ERDF for the availability of data used to carry out this project.

To my friends, I have to thank for fellowship, companionship, being there for me throughout the years and making college one of the best periods of my life. Special mentions go to: Luis Costa, Hugo Drumond, Guilherme Routar, Alexandre Ribeiro, Artur Alves, Jonas Antunes e Zé Miguel Mendes.

Lastly, the biggest gratitude and debt is to my family for their love, support, encouragement and comprehension not only in the last few months but during my whole life. To my parents I owe them everything and I am what I am and always will be because of them. A big thanks to my brothers whom I love and are the reason that kept me going. To my girlfriend, I have to thank for all the love, support, patience and for gifting me with one of the greatest blessings in this world, parenthood.

João Loureiro

"Insanity laughs under pressure we're cracking
Can't we give ourselves one more chance
Why can't we give love that one more chance
Why can't we give love
Cause love's such an old fashioned word
and love dares you to care for
The people on the edge of the night
And love dares you to change our way of
Caring about ourselves
This is our last dance
This is our last dance
This is ourselves
Under pressure"

Freddie Mercury

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
BP	Backpropagation
CDSS	Clinical decision support system
CV	Cross Validation
DL	Deep Learning
DNA	Deoxyribonucleic acid
DSS	Decision Support System
DT	Decision Tree
IS	Information Systems
MIS	Management Information Systems
ML	Machine Learning
MLP	Multi-Layer Perceptron
NN	Neural Network
RF	Random Forest
RNA	Ribonucleic Acid
SVM	Support Vector Machines

Chapter 1

Introduction

In a recent past a large number of innovations, breakthroughs and discoveries modeled the fields of Medicine and Biology. In even more recent days we are witnessing the rise of the Information Age, that brought the power of information and technology together.

The knowledge grasped from the medical and biological domains coupled with the large amount of data available and nowadays computational power enabled the creation of techniques and methods invaluable for humans health care. This is a mean to reach other goals, such as, the development of new procedures and treatments, computer-aided diagnosis, among others.

1.1 Context

Cancer is a group of diseases involving abnormal cell growth with the potential to invade or spread to other parts of the body, cancer has caused 9.6 million deaths in 2018 alone [1]. This contrasts with benign tumors, which do not spread to other parts of the body. Possible signs and symptoms can vary a lot from person to person but usually include one of the following: a lump, abnormal bleeding, prolonged cough, unexplained weight loss and a change in bowel movements. While these symptoms may indicate cancer, they may have other causes [2]. Humans are affected by over 100 types of cancer [4].

Computer-aided detection, also called computer-aided diagnosis, are systems that assist doctors in the interpretation of medical exams. Medical diagnostics can yield a great deal of information that the medical professionals has to analyze and evaluate comprehensively in a short time. Computer-aided diagnosis systems can output diagnostic results, such as possible diseases, in order to offer input to support a decision taken by the professional.

Computer-aided diagnosis can profit a lot with the use of Machine Learning. Machine Learning is a sub-discipline of Artificial Intelligence that uses algorithms and statistics to empower computers with learning skills. These models can be trained in exam data to be able to output a diagnosis on a new case. The diagnostic predicted must be always verified by a qualified doctor.

1.2 Motivation

With the increased volume and diversity of highly complex, multidimensional, relational and often weakly-structured and noisy data [26] [38] there is a need to design a system to automatically analyze it.

Genomics is an interdisciplinary field of biology that focus on the study of genomes structure, function, evolution, mapping, and editing. With the amount of data available about human DNA genomic research, scientists and clinicians are nowadays applying this knowledge to study the role that multiple genetic factors have on complex diseases, such as cancer, diabetes, and cardiovascular disorders.

When conducting clinical trials, scientists have the need to use tools that allow them to go from notes and experiment results to a final report. Leveraging the use of Decision Support Systems, Machine Learning and Data Mining that process can be improved upon.

Nowadays geneticists do not have a structured framework that allows them to push clinical trials raw original data to an infrastructure, explore the information in a easy to grasp and visual way and export datasets in order to create models, using ML algorithms, that can help solve problems by making predictions when fed with new data.

1.3 Goals

The goal of this project is to design and implement a framework that allows the application of Machine Learning to the analysis of genomic data. This framework must be able to deal with data from different types of analysis and experiments, allowing the scientist to process, organize and visualize the data, build Machine Learning models trained using that data, evaluate the results and finally produce a report like document with the entire process.

1.4 Dissertation Structure

Apart from this introductory chapter, the report has five additional chapters. Chapter 2 contains a detailed overview on the state of the art related to the project scope: Genomics, Decision Support Systems and Machine Learning. The history and current status of the fields related to this project's objective are described. Chapter 3 is a detailed and deeper overview of our problem, there we discuss the current issues that this project aims to solve, the proposed solution with its features and expected contributions. Chapter 4 describes the work that has been done, explains all of the implementation details and how various components were put together to reach the final goal. Chapter 5 describes the validation process, evaluation methods that were used and the results achieved. Lastly, Chapter 6 concludes the thesis work, presents its main difficulties and future goals.

Chapter 2

Literature Review

The goal of this Literature Review is to collect documents in the areas of Decision Support Systems, Machine Learning and Ensemble Learning to solve problems using genomic data. In order to reach our goal we used a systematic structured technique.

2.1 Genomics

Deoxyribonucleic acid (DNA) [5] is the key molecule in every living organism that contains the instructions needed to develop and direct the activities concerning each individual such as: functioning, growth, reproduction, and development. DNA molecules are made of two strands that coil around each other to form a double helix carrying the genetic instructions (Figure 2.1). There are four bases that can be present on DNA: Adenine(A), Cytosine(C), Guanine(G) and Thymine(T).

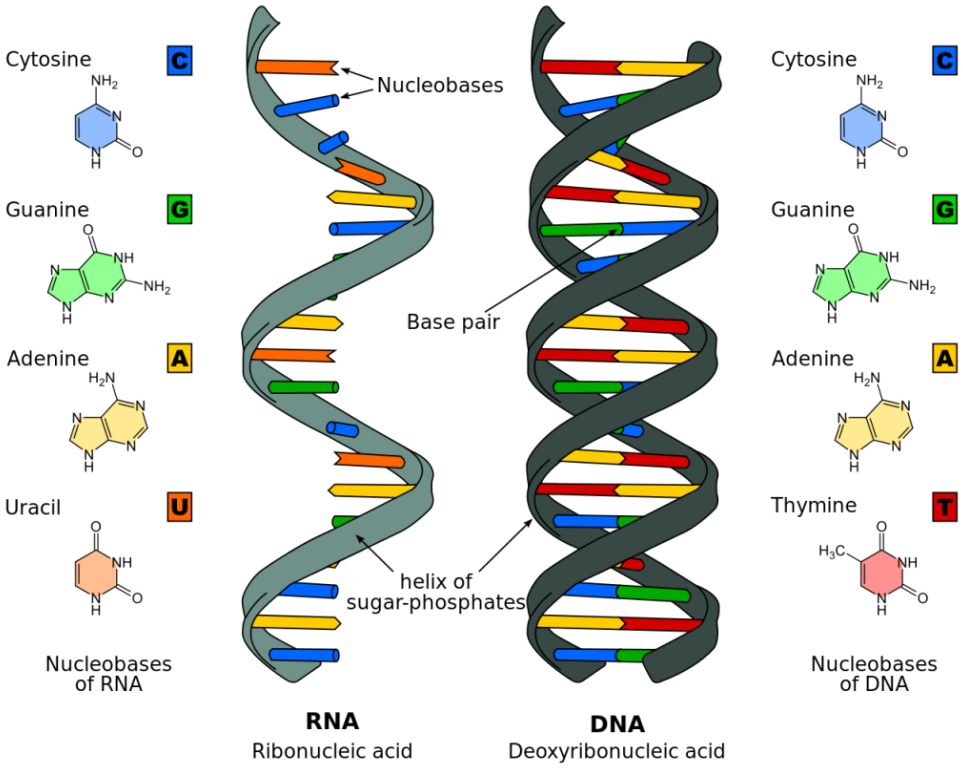


Figure 2.1: The Differences Between DNA and RNA [1].

Bases on opposite strands have pairing rules, meaning that an A always pairs with a T and a C always pairs with a G. The ordered sequence of bases is the representation of the information encoded in the DNA molecule, the same way a sequence of zeros and ones represents a program in binary code.

RNA stands for Ribonucleic Acid and is a molecule responsible for the coding, decoding, regulation and expression of genes. [13] RNA is just a single strand that folds onto itself(Figure 2.1). It is composed of four types of bases: Adenine(A), Cytosine(C), Guanine(G) and Uracil(U).

A genome is the complete set of DNA that belongs to an organism. The human genome for instance has approximately 3 billion DNA base pairs imprinted on every cell in the body. With its four-letter language, DNA contains the information needed to build the entire human body.

A gene often refers to a sequence of DNA that has the instructions needed in order to make a specific protein or set of proteins. They are contained on 23 pairs of chromosomes engraved into the nucleus of a human cell, genes dictate the production of proteins with the assistance of enzymes and messenger molecules. Gene expression is the process by which the information from a gene is used in the synthesis of proteins [14] and it is responsible for the regulation of cell differentiation. [43]

[1] https://www.thoughtco.com/dna-versus-rna-608191

Thus, the proteins created are responsible for the normal functioning of the body. If a cell contains a mutation in its DNA, it may not produce a specific protein or create an abnormal protein potentially having undesired effects that might lead to a disease such as cancer.

Genomics is a field of biology that focus on the study of genomes structure, function, evolution, mapping, and editing. With the amount of data available about human DNA genomic research, scientists and clinicians are nowadays applying this knowledge to study the role that multiple genetic factors have on complex diseases, such as cancer, diabetes, and cardiovascular disorders.

2.2 Decision Support System

A decision support system [29] (DSS) is an information system that supports the decision-making process, either by fully automating the task or generating valuable data that will help the users to make informed decisions. DSSs acts as a function that transforms massive reams of data in compiled information that can be used to solve problems and make better decisions. They are useful in problem-solving tasks, providing efficiency, speed and a more informed decision-making. From planning to management, these systems help persons to make decisions based on the problem data that can be unstructured, unorganized, big and incomprehensive by the end user.

Ralph H. Sprague, Jr. [49] in 1980 said that a DSS has the following characteristics:

- tend to be aimed at the less well structured, under-specified problem that upper level managers typically face;
- attempt to combine the use of models or analytic techniques with traditional data access and retrieval functions;
- focus on features which make them easy to use by non-computer-proficient people in an interactive mode
- emphasizes flexibility and adaptability to accommodate changes in the environment and the decision making approach of the user.

2.2.1 Background

In the late 1950s and early 1960s researcher tried to formulate the idea of computerized quantitative models to help in decision making and planning [28], but only in the 1960s the theory became a reality. Ferguson and Jones in 1969 produced the first reported experiment using a computer aided decision system [41], they used an IBM 7094 to run an application that could generate production schedules.

J.C.R. Licklider wrote a paper in 1960 called *Man-Computer Symbiosis* [34] where he explained why the interaction between man and machine would improve the human ability to solve problems making the process more efficient and having better results. His paper was a reference and influenced the Computer Science research for years to come.

With the development of systems with more computational power like IBM System 360 in 1967, the era of Information Systems (IS) began. These systems were powerful enough to allow big companies to implement Management Information Systems (MIS) with realistic and cost efficient requirements. Early MIS focused on accounting and transaction data and presented managers with periodic structured reports, but they did not support user interaction for decision making.

The concept of DSS was researched by many universities and important organizations by the 1980s. The results allow the idea to grow and conquer space being, by that time, an area of research of its own. The scope of DSS applications kept growing and gone beyond transfers and management, it was recognized that it could be implemented to support decision making in the multiple layers of an organization, *e.g.* decisions about: strategic, operations, financial, among other. By the end of 1980s the overlap between DSS and Artificial Intelligence (AI) became clear when the new challenge of building intelligent workstations appeared. [47]

In 1987, Texas Instruments designed a DSS that helped United Airlines to save time with the suggestion of new and more efficient management of ground operations in the airport. Gate Assignment Display System (GADS) was responsible for reducing travel delays in the airports that were using it. Beginning with the 1990s, data storage infrastructures and analytical processing began broadening the realm of DSS. With the introduction of the Word-wide web new Web-based analytical applications appeared and new opportunities for DSS rose.

The advent of more and better reporting technologies has seen DSS start to emerge as a critical component of management design. Examples of this can be seen in the intense amount of discussion of DSS in the education environment.

2.2.2 DSS Applications

DSS can theoretically be implemented in any knowledge domain.

Clinical decision support system (CDSS) are one of the applications of a DSS applied to health care science. CDSS can help doctors reach a diagnosis, choose a treatment and help in any other decision making problem that they face. The number of studies about CDSSs increased significantly since 1973. [21]

From the most common areas where DSS are applied one realy stands out: business and management. Dashboards are a prime example of the application of a DSS, thought it managers and executives can identify negative balances, reduce costs, better allocate resources, and others. All those improvements are done quicker and more precisely than it would happen without a DSS. With DSS all of a business information can be represented in the form of graphs, charts, tables, *i.e.* in a structured summarized fashion which translates into a high level view of the business.

DSS is extensively used in business and management. Executive dashboard and other business performance software allow faster decision making, identification of negative trends, and better allocation of business resources. Due to DSS all the information from any organization is represented in the form of charts, graphs i.e. in a summarized way, which helps the management to take strategic decision. One application of a DSS in management that it is not that obvious is the development of complex anti-terrorism systems.

DSSs cover a wide range of problem solving tasks in different areas. A few examples of the application of DSSs are: risk analysis, stock handling, sport bets, transfers in the financial sector; air lines, railways, forest, agriculture, business operations and strategies that all relate to management; and an infinite number of problems that can be represented in a model.

A specific example concerning the implementation of DSS in Corporate Functional Management [17]. The main areas that a DSS can be applied in this context are the following:

- Accounting/Auditing;
- Finance;
- Human Resources Management;
- International Business;
- Information Systems;
- Marketing;
- Production and Operations Management;
- Strategic Management;
- Multifunctional Management.

All these areas belong to just one specific field, that showcases the potential of DSSs to help humans in decision making on a wide range of problems.

2.3 Machine Learning

Artificial intelligence (AI) is one of the many fields of computer science. Machine learning (ML) is sub-discipline of AI that uses algorithms and statistic models to allow the ability of learning to computers without being programmed explicitly. A model is a simplified representation of reality. Arthur Samuel (1901–1990) was a pioneer of artificial intelligence research that first coined the term *Machine Learning* back in 1959. ML focuses on the development of computer programs that can access data and use it learn for themselves.

ML algorithms use sample data to build mathematical models, these models are a mathematical approximation of a objective function to what it is known as the data signal. The result of that approximation allows the model to make predictions or decisions without being explicitly programmed to perform the task.[9] Machine learning algorithms are used in various areas including but not exclusively: marketing, user profiling, email filtering, computer vision, medical diagnosis where it is impractical to implement an algorithm composed of specific instructions for performing the task.

The learning process begins with the *training data* that can be: observations, direct experience, or instructions. Models are fit to the training data in order to look for patterns and produce a

generalization that can make predictions in the future based on the examples from the training data. With that generalization from reality represented in the form of a model, computers can learn automatically without human explicit programming or assistance and adjust actions accordingly.

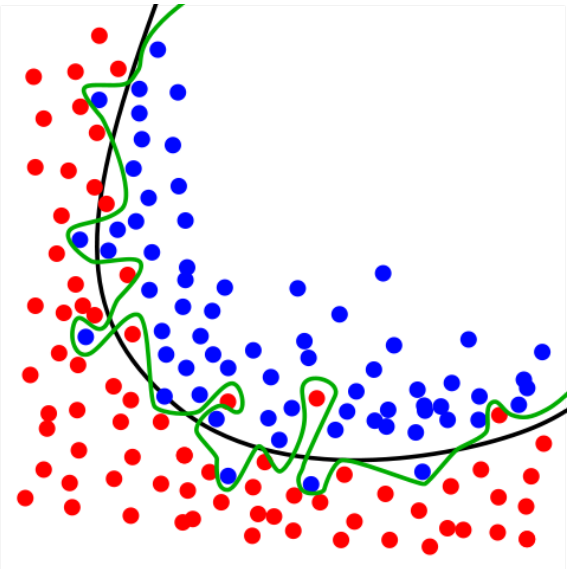


Figure 2.2: Representation of data signal and the objective function in training data [2].

In Figure 2.2 dots represent data that a model will be trained on. There are two different classes of objects in the picture, blue and red. The green line is a function that corresponds to the data signal, it perfectly separates the two classes. If a model objective function was exactly the same as the green line the model would have a perfect accuracy on this particular training set, the problem with that is the behaviour of the model with new and unknown data. The goal is to have an objective function that is closer to the black line, it ignores the noise on the data and represents a better generalization of the reality thus producing better predictions with new data.

2.3.1 Background

As mentioned above Arthur Samuel was the first to coin the term *Machine Learning* in 1959 while at IBM where he created a program that could play the famous checkers game. Since the early age of computers algorithms were developed that enabled modeling and analyzing sets of data. From the very beginning three major branches of machine learning emerged. Classical work in symbolic learning by Hunt [27] in 1966, in statistical methods by Nilsson [37] in 1965 and in neural networks by Rosenblatt [44] in 1958. Through the years advanced statistical and pattern recognition methods, such as Bayesian classifiers [20], the k-nearest neighbours [30], discriminant analysis [35], decision trees [11], artificial neural networks [45]. [32]

[2] https://github.com/justmarkham/scikit-learn-videos/blob/master/05_model_evaluation.ipynb

Figure 2.3: Arthur Samuel plays checkers with an IBM 704 computer in Poughkeepsie, New York

Arthur Samuel with his checkers playing program opposed the assumptions in that time that machines cannot go further than running written programs. He proved that computers could detect and learn from patterns like people do. His program was capable of observing positions in the game and learn a model that would yield better next moves, Samuel also noticed that his model was getting better over time with the increase of played games, *i.e.* observations.

He coined *Machine Learning* which he defines as:

"A field of study that gives computer the ability without being explicitly programmed."

Figure 2.4: Frank Rosenblatt

In 1958, Frank Rosenblatt's Perceptron[44] proposed a model based in his neuroscientific background that is similar to today's ML models. It was a very exciting discovery at that time, Rosenblatt defined his Perceptron, illustrated in Figure 2.5, with the following sentence:

> "The perceptron is designed to illustrate some of the fundamental properties of intelligent systems in general, without becoming too deeply enmeshed in the special, and frequently unknown, conditions which hold for particular biological organisms. [44, p. 387]"

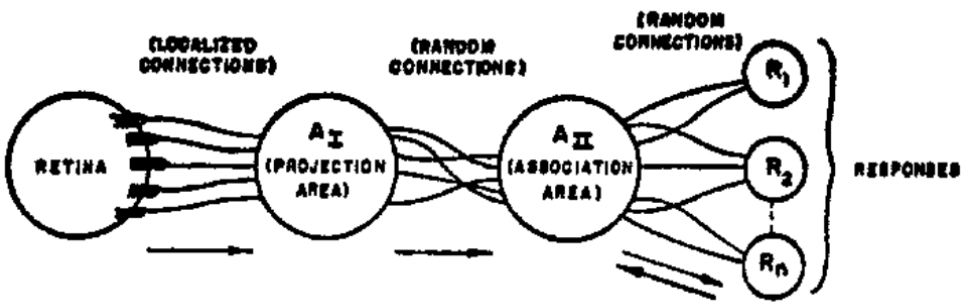


Figure 2.5: Organization of a perceptron [3].

However, Frank Rosenblatt's Perceptron hype was hinged by Minsky [36] in 1969. He proposed the famous XOR problem and the inability of Rosenblatt's solution to tackle non linearly separable data distribution. As Figure 2.6 shows in the output of the OR problem data can be split between the two classes, blue and black, by a single line *i.e.* it is a linear problem. But it is not the same case in the output of the XOR problem, where a single line is not enough to separete both classes. The solution proposed by Minsky was to combine Percepton units responses applyed to this problem in a second layer of units. It was the Minsky's tackle to Neural Networks (NN) community. Thereafter, NN researches would be dormant up until 1980s.

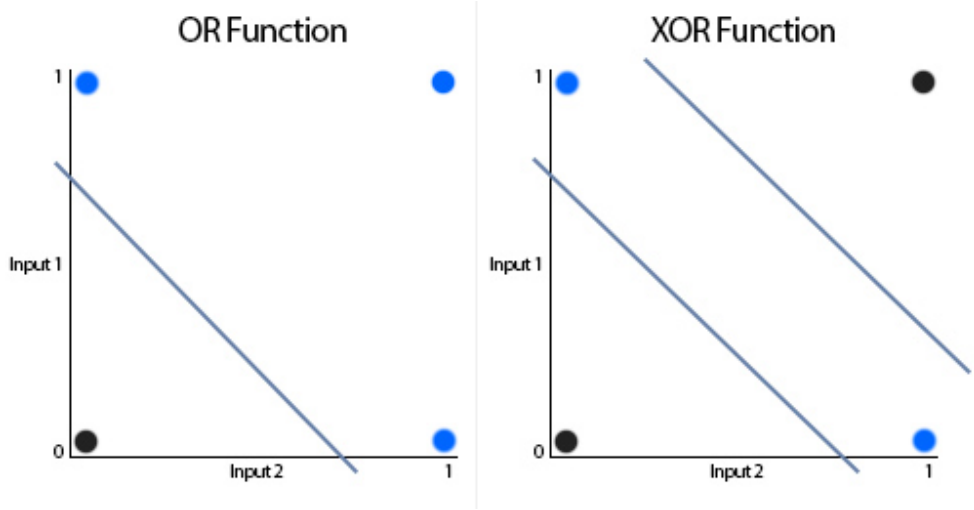


Figure 2.6: The OR and XOR problems [4].

In 1981, Werbos introduced the concept of Multi-Layer Perceptron [50] with NN first specific Backpropagation (BP) algorithm. BP is a key feature on the NNs of today, and Werbos paper ignited the research on the NN field and by 1986 researches successively presented the idea of MLP with practical BP training. (Rumelhart, Hinton and Williams [46] - Hetch-Nielsen [24])

[3] [44, Daniel J Power. A brief history of decision support systems.]
[4] https://djcordhose.github.io/machine-learning-in-the-browser/2016_rmjs.html

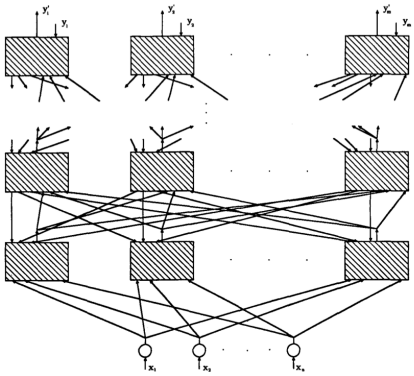


Figure 2.7: Hetch-Nielsen MLP [5].

With a different approach to ML, a very-well known algorithm was proposed by J. R. Quinlan [42] in 1986 known as Decision Trees (Figure 2.8), more specifically ID3 algorithm. Moreover, on the contrary of its NN models counterparts ID3 was released as a software able to find more real-life use case with its simplistic rules and its clear inference. After ID3, many different alternatives or improvements have been explored by the community, *e.g.* ID4, Regression Trees, CART and it is still one of the active topics in ML.

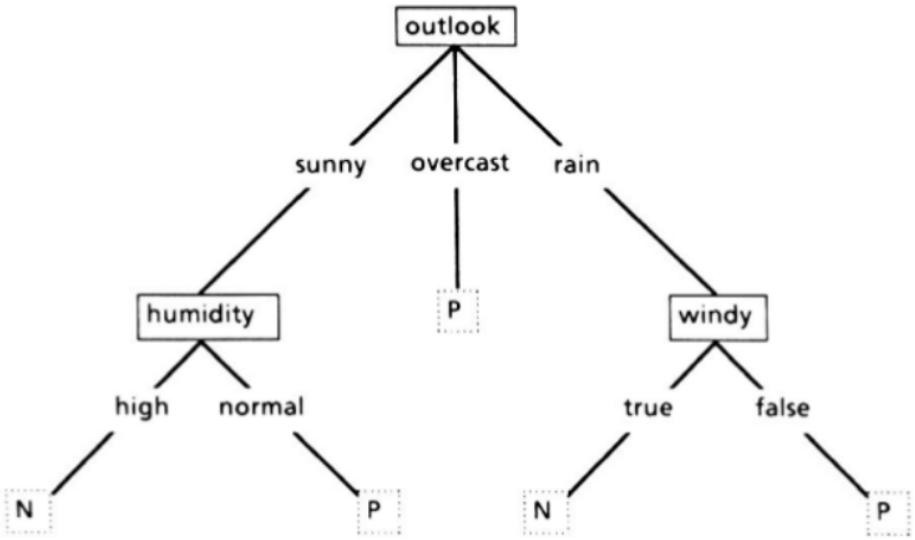


Figure 2.8: A simple decision tree [6].

One of the most important ML breakthroughs was Support Vector Machines (SVM), proposed by Vapnik and Cortes [15] in 1995 with a very strong theoretical standing and empirical results.

[5] [24, Robert Hecht-Nielsen. Theory of the backpropagation neural network.]
[6] [42, J. Ross Quinlan. Induction of decision trees.]

SVM was capable of applying the knownledge of convex optimization, generalization margin theory and kernels. Thus it was applicable in a wide range of fields and areas causing significant and extremely rapid improvements.

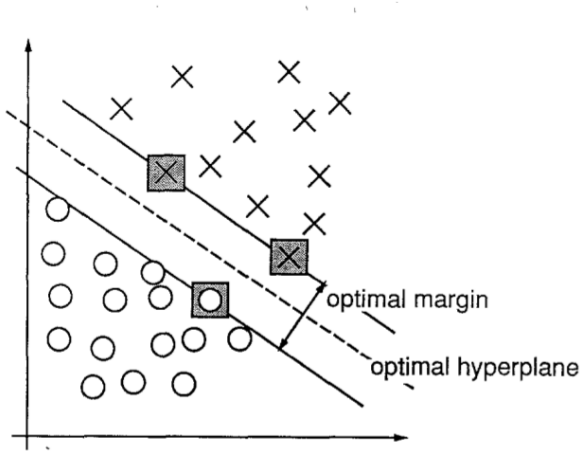


Figure 2.9: An example of a separable problem in a 2 dimensional space. The support vectors, marked with grey squares, define the margin of largest separation between the two classes [7].

By the late 1990s a trend was created to reuse previous known models and combine them using ensemble learning methods.

In 1997, a new and powerful ML model was proposed by Freund and Schapire that used boosting, an ensemble technique, of weak classifiers called Adaboost [18]. Adaboost trains weak set of classifiers that are easily and quickly trained, by giving more importance to hard instances. The weak classifiers are picked as single decision tree nodes. They introduced Adaboost as:

> "The model we study can be interpreted as a broad, abstract extension of the well-studied on-line prediction model to a general decision-theoretic setting" [18, Abstract]

Another model that leverages the use of ensemble was explored by Breiman [10] in 2001 it ensembles multiple decision trees, each created by a random subset of instances and its nodes are chosen from a random subset of features. Therefore, it is called Random Forests (RF). RF has theoretical and empirical proofs of endurance against over-fitting, due to its random nature. Even the above mentioned AdaBoost shows weakness against over-fitting and outliers in sample data, RF is a very fool proof model to these particular problems. RF shows its success in many different tasks, it is one of the most common algorithms in Kaggle[8] competitions.

> "Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges a.s. to a limit as the number of trees in the forest becomes large [10, Abstract]."

[7] [15, Corinna Cortes and Vladimir Vapnik. Support-vector networks.]
[8] https://www.kaggle.com/

Nowadays, a new era of NN called Deep Learning (DL) rose. Deep Learning takes the concept of NNs and extends it by creating a NN model composed by a predifined number of layers, tipically a huge number of layers. The DL era began in 2005 with the conjunction of many different discoveries from past and present researchers and it is a field of its own now.

2.3.2 ML Applications

Applications for machine learning include but are not limited to:

Advertising, agriculture, bioinformatics, computer networks, computer vision, data quality, economics, financial analysis, fraud detection, games, genomics, handwriting recognition, information gathering, insurance, linguistics, marketing, medical diagnosis, natural language processing and understanding, optimization, perception, recommender systems, robotics, search engines, sentiment analysis, speech recognition, telecommunication, theorem proving and translation.

The ultimate goal of ML is to train models with data that represent an approximation to reality in order to make predictions based on new input. Consider an example of medical diagnosis where a doctor, based on a series of exams, has to tell a patient that it suffers from a particular disease. A prediction in this example would be the probability of the patient having that disease, the model would need to learn from previous similar exams from other patients and make a prediction based on the exams of this particular patient.

There are two types of ML models based on the prediction that they can produce: classification and regression models.

In classification the dependent variables are categorical and unordered (*e.g.* 'pneumonia', 'common cold', 'tuberculosis'), it is called classification because each observation of the sample data belongs to a class. Returning to the medical diagnosis example, each patient is an observation, the exams results are features of that observation and the diagnosis is the class that that patient belongs to.

With regression the dependent variables are continuous or ordered whole values, typically real numbers. Regression estimates the conditional expectation of the dependent variable given the independent variables. A simple example of a regression model would be a model that based on a person height could determine their weight.

One of the most common uses of machine learning is computer vision. There are many real life situations where digital image needs to be classify. When people use their smartphones to take a picture most devices have the ability to detect faces, this is a typical computer vision classification problem where the model has to classify the image by having a face or not having a face. Some smartphones have the capability of unlocking when they detect the owners face this is also a classification problem, the owners face present in the image or no owners face.

For digital images, the features are extracted from the outputs of each pixel. In the case of a black and white images, the shade of each pixel serves as one measurement. In colored images, each pixel has more information about itself thus it represents more of a challenge and are a lot more interesting to work with.

Medical Diagnosis as stated before is one of fields where the application of ML models is present. It provides methods, techniques, and tools that can help solving diagnostic and prognostic problems in a variety of medical domains. It is being used for the analysis of the importance of clinical parameters and of their combinations for prognosis, *e.g.* prediction of disease progression, for the extraction of medical knowledge for outcomes research, for therapy planning and support, and for overall patient management. ML is also being used for data analysis, such as detection of regularities in the data by appropriately dealing with imperfect data, interpretation of continuous data used in the Intensive Care Unit, and for intelligent alarming resulting in effective and efficient monitoring.

Successful implementation of ML methods can help the integration of computer-based systems in the healthcare environment providing opportunities to facilitate and enhance the work of medical experts and ultimately to improve the efficiency and quality of medical care.

In medical diagnosis, the main interest is in establishing the existence of a disease followed by its accurate identification. There is a separate category for each disease under consideration and one category for cases where no disease is present. Here, machine learning improves the accuracy of medical diagnosis by analyzing data of patients.

The measurements in this application are typically the results of certain medical tests (*e.g.* blood pressure, temperature and various blood tests) or medical diagnostics (*e.g.* medical images), presence, absence and intensity of various symptoms and basic physical information about the patient such as age, sex and weight. On the basis of the results of these features, the doctors narrow down on the disease inflicting the patient.

2.3.3 Decision Trees

Decision Tree (DT) is a supervised learning method used to solve classification and regression problems that can handle numerical and categorical data. DTs are a tree like structure that represents a set of if-then-else decision rules, learned from training data. DT builds models that break down a data set into smaller and smaller subsets in each node. With each decision or break an associated decision tree is incrementally developed. In the end the tree created contains two types of nodes: decision nodes and leaf nodes. A decision node is basically an if-else statement, while the leaf node represents a prediction for the classification or regression problem. From top to bottom the importance of the decision nodes always decreases, being the root node the best predictor.

A DT is built from a root node to the leaf nodes and involves constant partitioning of the data into subsets that contain instances with homogeneous values, *i.e* instances from the same class. ID3 algorithm [42] was the first algorithm invented to produce a DT, it uses entropy to calculate the homogeneity of a sample data. If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.

Table 2.1: A small training set [9].

No.	Outlook	Temperature	Humidity	Windy	Class
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

Table 2.1 is the training set given in the paper Induction of Decision Trees by Quinlan in 1985 [42]. In order to extrapolate a DT from a training set the ID3 algorithm selects the best predictor from the available features. The best predictor is the feature that provides the best information gain, a metric defined in the paper by its author.

To understand information gain it is essential to first understand set purity. A set of data can be pure or impure, sets with just one class are pure and sets with more than one are impure. In the example training set, if we only take into account the set were the outlook feature is overcast it has 4 Ns and 0 Ps this exemplifies a pure set. On the contrary a set that contains all the sunny values of outlook has 3 Ns and 2 Ps which means this is an impure set. Furthermore, it is necessary to have a metric that can represent the notion of pure and impure sets. Note that a set that has 8 Ps and 0 Ns and a set that has 2 Ns and 0 Ps are both pure, so the probability of P or N can not be used and this is were it is used the entropy.

The information required for classification given a dataset is calculated by the following equation:

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n} \tag{2.1}$$

The information required for classification given that the attribute chosen for the root of the DT is A, *i.e.* A entropy is calculated by the following equation:

$$E(A) = \sum_{i=1}^{v}\frac{p_i+n_i}{p+n}I(p,n) \tag{2.2}$$

[9] [42, J. Ross Quinlan. Induction of decision trees.]

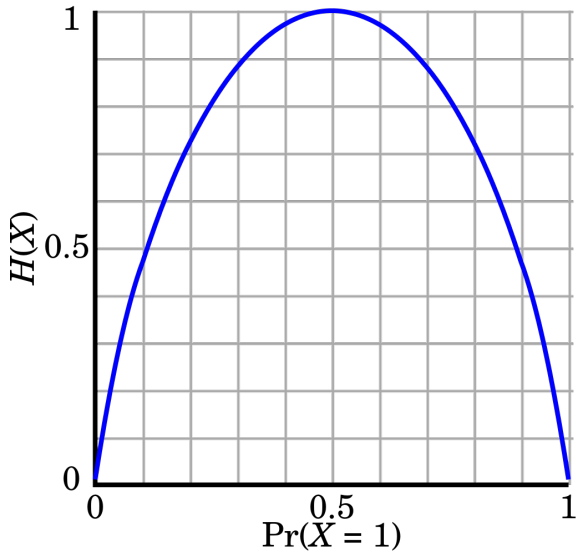


Figure 2.10: Entropy function [10].

Figure 2.10 illustrates the reason to use entropy to determinate set purity. If a set is pure its entropy is 0 and if a set is impure having half of the values from one class and the other half from another class, *i.e.* the worst case scenario the entropy is 1.

Using the equations from information and entropy above mentioned it is possible to calculate the information gain of choosing A as the DT root by subtracting A entropy to the information required to classify the entire dataset:

$$gain(A) = I(p,n) - E(A) \tag{2.3}$$

A good rule of thumb would be to choose that attribute to branch on which gains the most information. [42]

To illustrate the idea, consider the set of objects in Table 2.1. Of the 14 objects, 9 are of class P and 5 are of class N, so the information required for classification is:

$$I(p,n) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940bits \tag{2.4}$$

Using the outlook attribute as an example to calculate the entropy. It has three values: sunny, overcast and rain. Five of the 14 objects have the value sunny, two of them from class P and three from class N. So the information required for classification given the subset were we only consider sunny objects is:

$$p_1 = 2, n_1 = 3 \tag{2.5}$$

$$I(p_1,n_1) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971bits \tag{2.6}$$

[10] https://en.wikipedia.org/wiki/Information_theory

Similarly for overcast:

$$p_2 = 4, n_2 = 0 \tag{2.7}$$

$$I(p_2, n_2) = 0 bits \tag{2.8}$$

And for rain:

$$p_3 = 3, n_3 = 2 \tag{2.9}$$

$$I(p_3, n_3) = 0.971 bits \tag{2.10}$$

The expected information requirement for outlook is therefore:

$$E(outlook) = \frac{5}{14} I(p_1, n_1) + \frac{4}{14} I(p_2, n_2) + \frac{5}{14} I(p_3, n_3) = 0.694 bits \tag{2.11}$$

The gain of information is then:

$$gain(outlook) = I(p, n) - E(outlook) = 0.246 bits \tag{2.12}$$

Applying the same process for the rest of the attributes the results are the following:

$$gain(temperature) = 0.029 bits \tag{2.13}$$

$$gain(humidity) = 0.151 bits \tag{2.14}$$

$$gain(windy) = 0.048 bits \tag{2.15}$$

In this situation ID3 would choose the outlook attribute since it has the higher information gain value. The objects would then be divided into subsets according to their values of the outlook attribute Table 2.2 is the subset of objects that the value of outlook is sunny and a decision tree for each subset would be induced in a similar fashion. In fact, Figure 2.11 shows the actual decision tree generated by ID3 from this training set.

Table 2.2: The subset of objects where the outlook is sunny.

No.	**Temperature**	**Humidity**	**Windy**	**Class**
1	hot	high	false	N
2	hot	high	true	N
3	mild	high	false	N
4	cool	normal	false	P
5	mild	normal	true	P

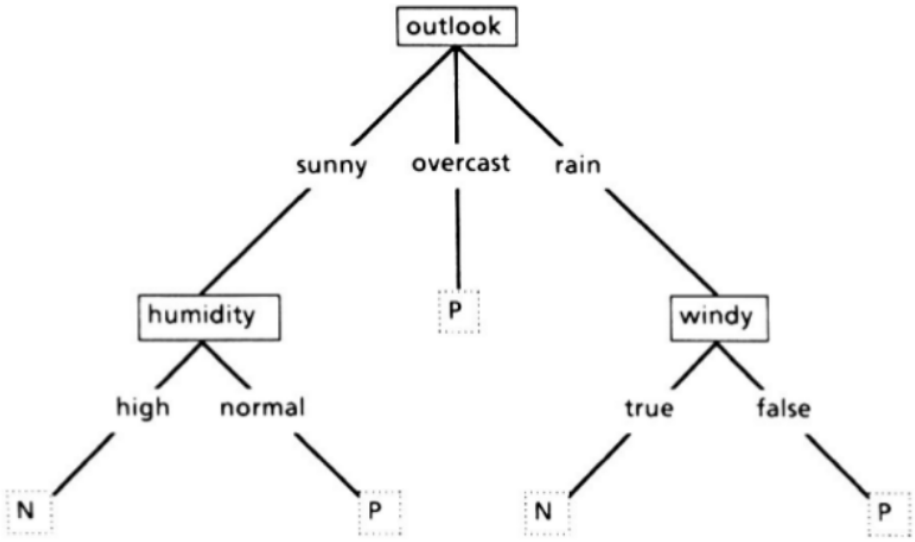


Figure 2.11: ID3 DT generated from Table 2.1 training data [11].

Another metrics were develop over time that help the generation of DTs, gini index [23] is another good example similar to information gain.

2.3.3.1 Random Forest

> "Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges a.s. to a limit as the number of trees in the forest becomes large [10, Abstract]."

Decision trees are easy to build, with low computational power requirements, easy to use and easy to interpret, but in practice they rarely have a good representation of the data signal.

> "Trees have one aspect that prevents them from being the ideal tool for predictive learning, namely accuracy [19]."

In other words, they work great with the data used to create them, but they are not flexible when it comes to classifying new samples. Random Forest combines the simplicity of decision trees with flexibility, resulting in a vast improvement in accuracy. [19, p. 587–588]

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging. The first algorithm for random decision forests was created by Tin Kam Ho [25] using the random subspace method [7].

A Random Forest can be constructed using the following algorithm:

[11] [42, J. Ross Quinlan. Induction of decision trees.]

1. Let the number of training objects be N and the number of features in the training data be D.

2. Choose L to be the number of individual forests in the tree.

3. For each individual tree l, choose nl (nl < N) to be the number of objects to train l with. Select randomly the nl objects, it is common to have repeated objects from the original data.

4. In the training of model l, each time the tree selects a node only dl randomly selected features from D are used.

At the end of the algorithm, the Random Forest model is composed of L Decision Trees. In order to use the model to make a prediction on new data, each of the trees from the forest outputs its own prediction and usually the prediction that appears the most is the one the Random Forest model will choose, pretty much like a voting system.

2.3.4 Support Vector Machines

> "The support-vector network is a new learning machine for two-group classification problems. The machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high- dimension feature space. In this feature space a linear decision surface is constructed [15, Abstract]."

In machine learning, support-vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

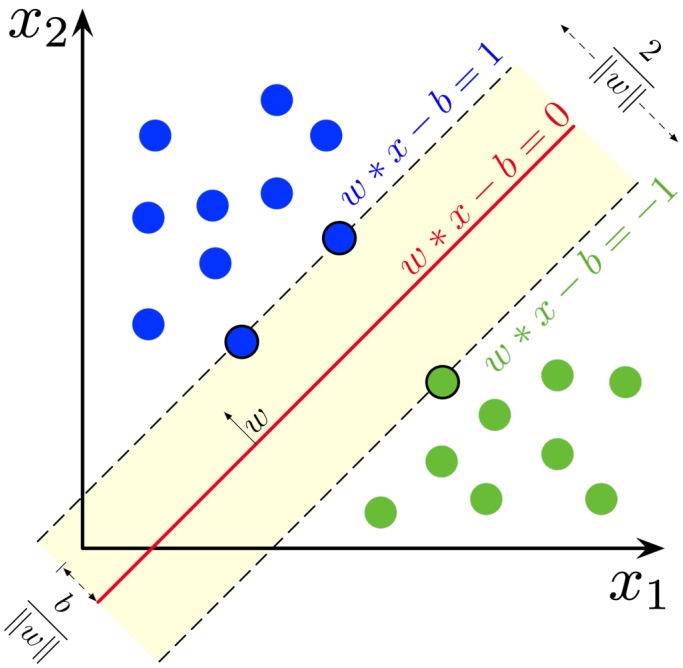


Figure 2.12: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors [12].

Figure 2.12 illustrates the representation of objects that can belong to two different classes, blue and green. The dashed line is the hyperplane computed by the SVM model to separate the two classes, its equation is the result of an optimization function that maximizes the gap between classes. The gap is the sum of the distances between the hyperplane and the nearest object for each class, those objects are called vectors and in this figure we have three, two blues and one green. This example is reduced to two dimensions because it is simpler to represent and understand, but real applications of SVMs have extremely high dimensions.

Classifiers learn from high dimensional features spaces, without actually having to map the points into the high dimensional space. Furthermore, data may be linearly separable in the high dimensional space but not linearly separable in the original feature space and there is where the application of kernels can be used to leverage the fact that they are not tied to the SVM formalism and can also be applyed to objects that are not vectors.

2.3.5 Ensemble Learning

Ensemble learning is a machine learning paradigm that combines multiple models to solve a problem. In contrast to ordinary machine learning approaches which try to learn one hypothesis from training data, ensemble methods combine a set of hypotheses.

Supervised learning algorithms perform the job of searching through a hypothesis space to find a suitable hypothesis that will make predictions given a particular problem. Even if the hypothesis

[12] https://en.wikipedia.org/wiki/Support-vector_machine

space contains hypotheses that are weak for a particular problem. Ensembles combine multiple hypotheses to form an improved hypothesis.

An ensemble is composed by a set of learners frequently called base learners, because it represents the aggregation of models it usually provides a much stronger and generalized prediction. One of the advantages of using ensemble learning is that it is able to boost weak models by combining them, some of these models can be overfitting the data and others only be accurate in a particular subset and the generalization of the ensemble method brings the best of both worlds and comes out as a stronger model. [40]

Ensemble methods, Figure 2.13, are a supervised learning algorithm, they are trained in labeled sample data and then make predictions. The ensemble model represents a single hypothesis that may or may not be contained in the space of hypothesis from the base learners used to built it. For that reason, ensembles are more versatile in the functions that they can represent. In theory, make the ensemble model more prone to overfitting , but in practice usually ensemble techniques tend to reduce problems related to overfitting of the training data. [48]

Any kind of ML algorithm can be a base learner, even ensemble models can be used as a base learner for other ensemble models. The same way that Random Forest only uses Decision Trees, ensemble methods tend to use a single base learning algorithm to produce homogeneous base learners, but there are also some methods which use multiple learning algorithms to produce heterogeneous learners.

Empirically, ensembles tend to yield better results when there is a significant diversity among the models. [33] [48] Many ensemble methods, therefore, seek to promote diversity among the models they combine. Thus, random algorithms can be used to produce a stronger ensemble than very explicit programmed algorithms.

Evaluation of an ensemble method predictions usually requires more computational power than evaluating the prediction of a single model, so ensembles may be thought of as a way to compensate for poor learning algorithms by performing a lot of extra computation. Fast algorithms such as decision trees are commonly used in ensemble methods, *e.g.* random forests. [10]

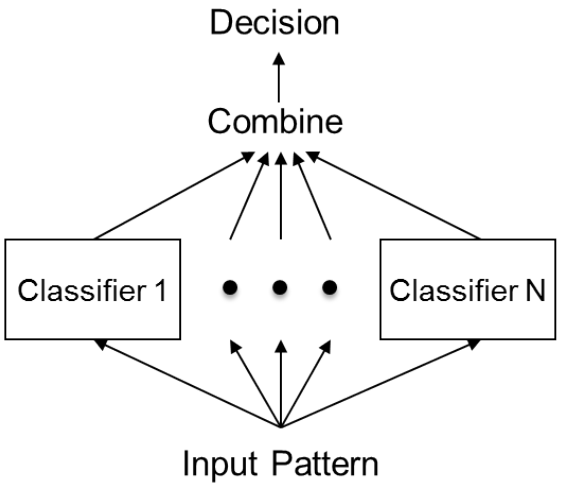


Figure 2.13: High level view of an ensemble method [13].

2.3.5.1 Stacking

> "Stacked generalization is a scheme for minimizing the generalization error rate of one or more generalizers. Stacked generalization works by deducing the biases of the generalizer(s) with respect to a provided learning set. This deduction proceeds by generalizing in a second space whose inputs are (for example) the guesses of the original generalizers when taught with part of the learning set and trying to guess the rest of it, and whose output is (for example) the correct guess [51, Abstract]."

Stacking or Stacked Generalization is an ensemble method that combines multiple models to output a prediction. It does so by exploring a space of different models for the same problem. To achieve that goal different models are trained in some part of the problem, but not the whole space.

[13] http://www.writeopinions.com/ensemble-learning

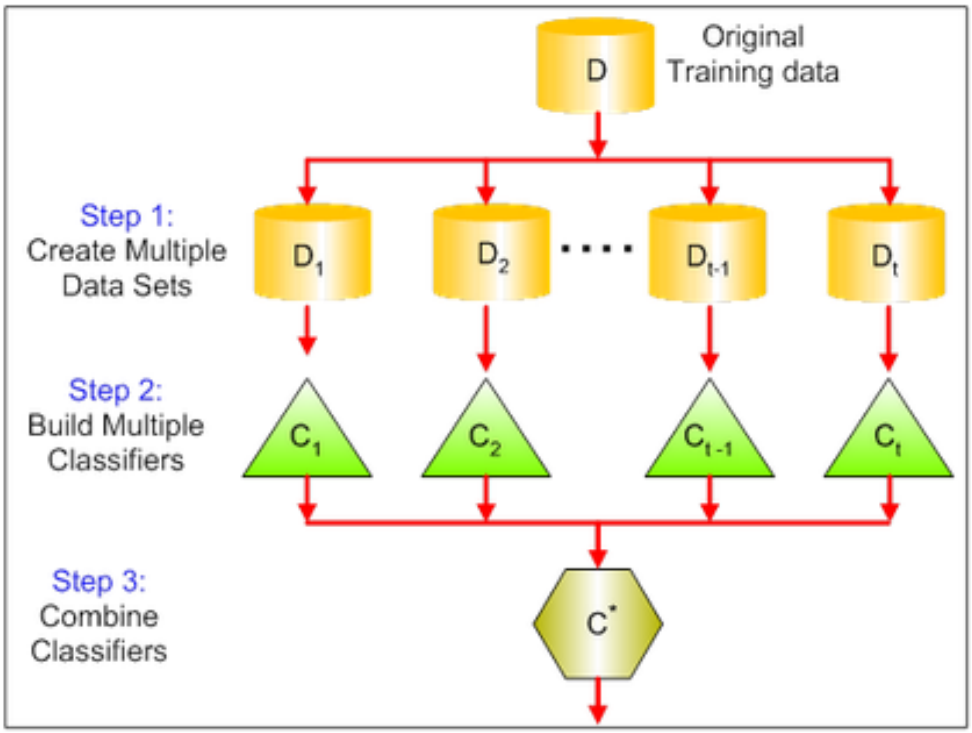


Figure 2.14: High level view of a Stacked Generalization [14].

Figure 2.14 illustrates the process to achieve a Stacked Generalization. The training dataset is divided in t parts, these parts can be copy of the original dataset or subsets of the original data. Multiple base learners are used them to build an intermediate prediction, one prediction for each learned model. Then a new model is trained with all the intermediate predictions from the base learners. The name stacking comes from the fact that the final model is stacked on the top of the base weaker models. Thus you might improve your overall performance, and often you end up with a model which is better than any individual intermediate model. As is often the case with any machine learning technique, there is no guarantee that the stacked model will perform better than any individual base model.

2.3.6 Cross Validation

Cross-validation also known as rotation estimator and out-of-sample testing is a technique that is used for the assessment of how the results of statistical analysis generalize to an independent data set. [22] [31]

The goal of cross-validation is to estimate how accurately a predictive model will perform in practice. In most cases, to solve a prediction problem a model is usually given a training dataset that contains the features of that problem that a model can learn from. The goal of cross-validation

[14] https://www.analyticsvidhya.com/blog/2015/09/questions-ensemble-modeling/

is to test the model's ability to predict new data, *i.e* a test dataset, that was not used in the training stage in order to identify common problems like overfitting or selection bias. [12]

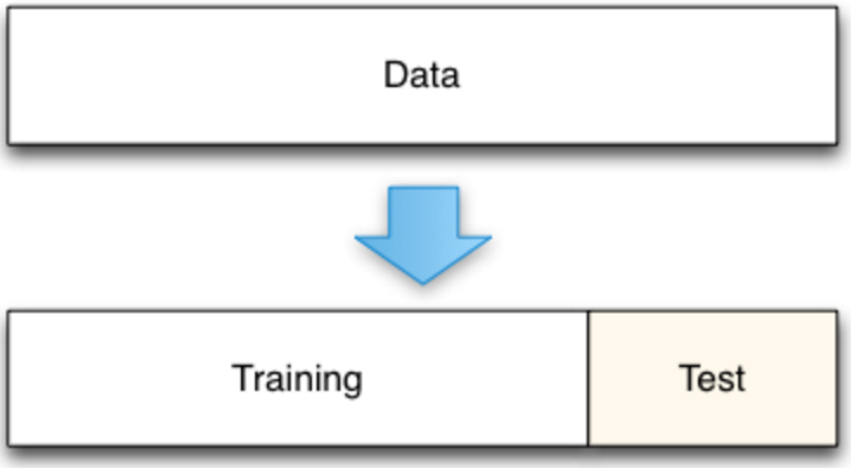


Figure 2.15: Data split in training and testing datasets [15]

As shown in Figure 2.15, cross-validation divides the sample data into complementary subsets, *i.e* training and test. Models use the training subset to learn from and validation is done using the test subset which the model never learned from, hence they are equivalent to new data.

In some of the most known methods, like K-fold, multiple iterations of cross-validation are performed using different partitions for training and test datasets and the validation results are combined , *e.g. averaged*, over the rounds to give an estimate of the model's predictive performance.

2.3.6.1 Holdout Method

> "The holdout method is the simplest kind of cross validation. The data set is separated into two sets, called the training set and the testing set. The function approximator fits a function using the training set only. Then the function approximator is asked to predict the output values for the data in the testing set (it has never seen these output values before). The errors it makes are accumulated as before to give the mean absolute test set error, which is used to evaluate the model. The advantage of this method is that it is usually preferable to the residual method and takes no longer to compute. However, its evaluation can have a high variance. The evaluation may depend heavily on which data points end up in the training set and which end up in

[15] http://scott.fortmann-roe.com/docs/MeasuringError.html

> the test set, and thus the evaluation may be significantly different depending on how the division is made." [3, Holdout Method]

In the holdout method, data is randomly split into two sets, training and testing. Typically the test set is considerably smaller than the training set (*eg. 70/30 split*), although the size of each of the sets is arbitrary. A model is then trained on the training set and the test set is used to evaluate it.

While the holdout method is usually defined as "the simplest kind of cross-validation" [3], many sources instead classify holdout as a type of simple validation, rather than a form of cross-validation. [31] [6]

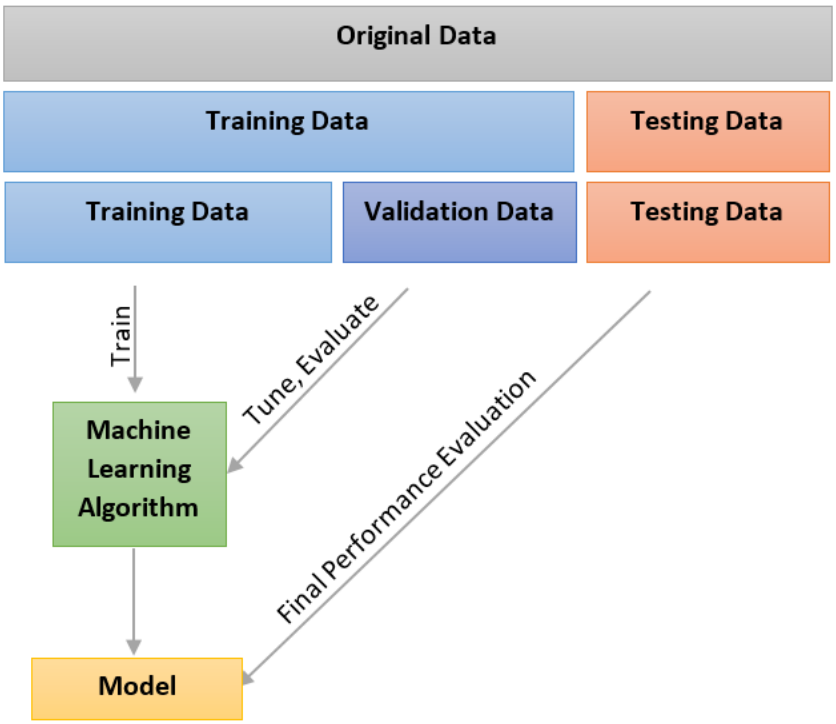


Figure 2.16: Hold-out method overview [16].

Figure 2.16 provides an overview of the Holdout method, with the addiction of a new set, the validation data. In this example the original data is divided into training data and test data, then the training data is also divided into training and validation sets. This allows a first evaluation of the model in order to tune its hyper parameters. Fiddling with the hyper parameters of a model algorithm using the evaluation provided by the test set would result in a form of overfitting since the changes would be implemented based on what it is supposed to be unknown data. After the train and tuning of the model the final performance is evaluated insuring that the test set acts as new data to the model.

[16] https://www.codeproject.com/Articles/1146582/Introduction-to-Machine-Learning

2.3.6.2 K-fold Cross Validation

> "K-fold cross validation is one way to improve over the holdout method. The data set is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set. Then the average error across all k trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times. The variance of the resulting estimate is reduced as k is increased. The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation. A variant of this method is to randomly divide the data into a test and training set k different times. The advantage of doing this is that you can independently choose how large each test set is and how many trials you average over [3, K-fold Cross Validation]."

In the K-fold Cross Validation method, data is randomly split into K subsets and the holdout method is repeated K times, each iteration uniquely uses one of the K subsets as the test set and the remainder K-1 subsets as the training set. The evaluation of each iteration is computed and the average represents the evaluation of the models.

Figure 2.17: K-fold Cross Validation method overview [17].

Figure 2.17 is an overview of the K-fold Cross Validation with an example that uses ten folds. The original training set is divided in ten equal parts, each of the parts is used once as the test data for evaluating the model, giving a total of ten iterations. The results of the evaluation of the models in each iteration are then summed and divided by the number of folds, this result is the overall evaluation of the models.

[17] https://medium.com/@sebastiannorena/some-model-tuning-methods-bfef3e6544f0

In stratified k-fold cross-validation, the folds are selected so that the mean response value is approximately equal in all the folds. In the case of binary classification, this means that each fold contains roughly the same proportions of the two types of class labels, based on the distribution of those classes in the original data.

Chapter 3

Problem Statement

The purpose of this chapter is to define the problem and state the issues encountered today, declare a proposal to mitigate this issues, consider the assumptions that were taken, enumerate the features and the use cases to implement. After those initial points, questions concerning to the research phase are laid out, the validation method explained and the results obtained discussed.

3.1 Current Issues

Nowadays geneticists don't have a structured framework that allows them to push clinical trials raw original data to an infrastructure, explore the information in a easy to grasp and visual way and export datasets in order to create models, using ML algorithms, that can help solve problems by making predictions when fed with new data.

There are no tools or frameworks that can deliver a complete workflow for the next set of tasks when working with genomic data:

- **Preprocessing** — Preprocessing raw data that is the result of a clinic trial;
- **Import** — Import the raw data to a database infrastructure;
- **View metadata** — Display information about trial, experiments within the trial (*eg. copy number analysis, gene expression z-score*);
- **View statistics** — Using graphs and tabular data display statistics about the attributes of each experiment on a clinical trial basis;
- **Export** — Export a dataset for the clinical trial on a specific format (*eg. CSV, ARFF*) based on the options passed by the user (*eg. experiments to include, type of validation strategy, use of ensemble learning*);
- **Processing** — Change the dataset in a convenient way for the use on ML algorithms;
- **Modeling** — Create a ML model that can predict the diagnosis based on the dataset chosen;

- **Evaluate** — Evaluate the models created;
- **Report** — Easy to view, report like document of that contains the *Processing*, *Modeling* and *Evaluate* steps.

3.2 Proposal

The goal of this project is to design and implement a framework that allows all the typical ML related workflow in a single multi-platform environment for the process and modeling of genomic data. Considering the tasks declared in Chapter 3.1, there can be grouped in two types of classes: computational and display, as Table 3.1 demonstrates.

Table 3.1: Tasks of the workflow divided into two groups.

Computational	**Display**
Preprocessing	View Statistics
Import	View Metadata
Processing	Export
Modeling	
Evaluate	
Report	

In order to perform all those tasks the framework will be a mixture of existing technologies with new custom made components that will live under the same environment making it easy to interact and maintain.

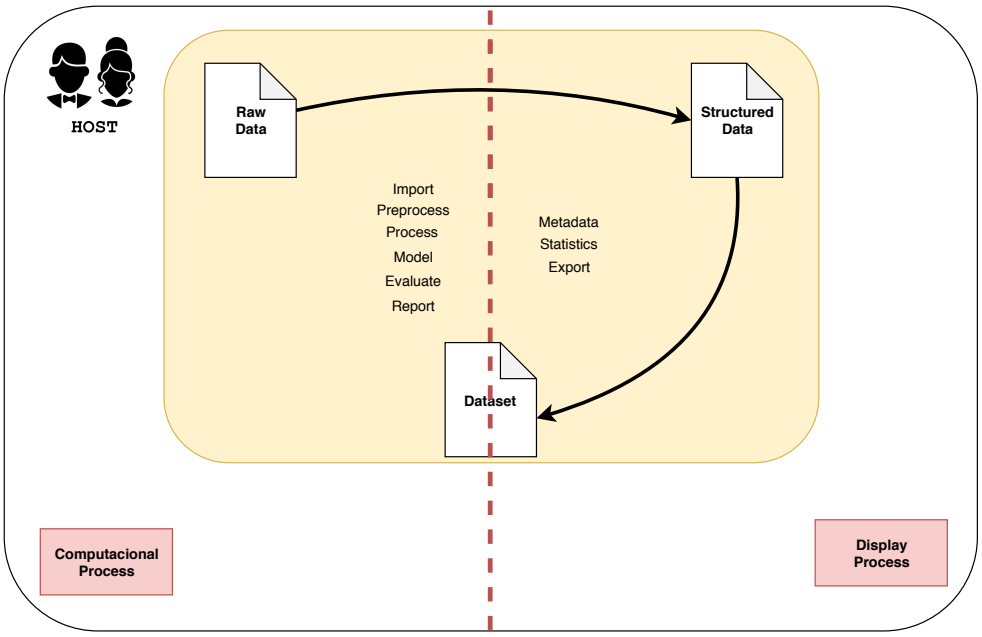


Figure 3.1: High level view of the proposal architecture and workflow.

Figure 4.3 shows a high level view of a framework requirements running on a single host . Computational tasks are responsible for transform data and store it in a predefined structure and for extracting knowledge from the dataset with modeling. Display tasks allow the user to grasp the contents of the structured data in a condensed and graphic visualization.

3.3 Assumptions

When performing a clinical trial or simply gathering data from already existing procedures, the raw data can be in an infinite number of formats or even have no format at all. The import phase of the raw data is entirely the responsibility of the user, it will be impractical to try to model every single format or to restrict the user to a couple formats.

Although the framework expects a certain structure within the data it can be very versatile, the only requirements being that each clinical trial has to be a database in *mongo* and that each experiment from that trial has to be a collection of that database.

Due to the high complexity of ML tasks after the export of the dataset, the user is able to create his own methods and run his own procedures to process the dataset, create models and evaluate them. These steps have to be done manually it is not the purpose of this project to automate the creation and evaluation of ML models.

3.4 Features

This section enumerates the features that the framework will cover. Although some of the features are intrinsic to the underlying technologies the overall combination of existing and new into a framework is what adds value to the present solutions.

In order to be more intuitive to showcase the features and to express the thought process behind, it is illustrated in Figure 3.2 a simple case of a geneticist that recently performed a clinical trial in several patients that are healthy or have stomach cancer. In this trial he conducted three genetic experiments: *gene expression*, *copy number analysis* and *methylation 450*.

The end goal of the geneticist is to create a model that can predict whether or not a patient has stomach cancer given any combination of these three genetic experiments.

To better showcase the use of the framework, the use cases presented below are in an order that is considered to be the best workflow from raw data to a report.

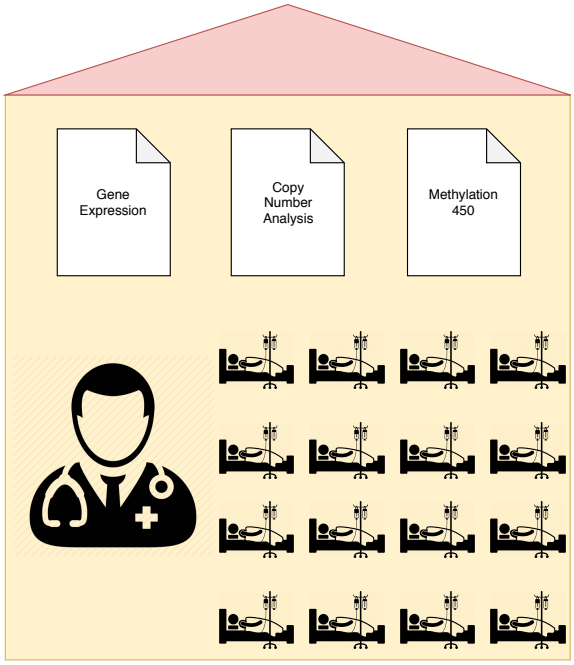


Figure 3.2: A possible use case scenario.

3.4.1 Use Case 1: Pre-process

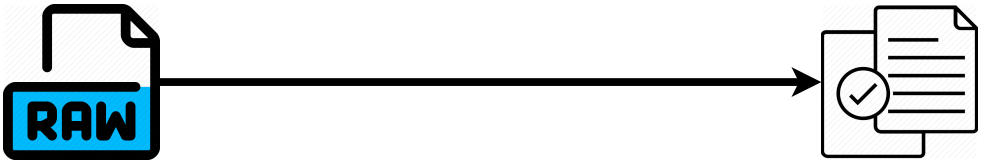


Figure 3.3: Preprocessing use case. Transform raw data into a data structure.

The framework must be able to allow the user to pre-process raw data. Raw data contains features, in the form of results, extracted from experiments conducted during the clinical trial. Thus it can be poorly organized, gathered without a standard format not well suited for storage in data structures used in computer science and it can contain a lot of unnecessary and meaningless information.

As an example for our scenario, figure 3.4 represents the top 10 lines of a *txt* file with the results of a Gene Expression experiment conducted on an individual that has stomach cancer, each patient has its own file. In order to identify the sample given by this patient these lines have an unique identifier called *GDC_Aliquot* that appears in every single line of the file, a lot of redundancy here since each patient has its own file. Next we have the chromosome being tested by sections or genes, in the excerpt we have 9 sections of the first chromosome and the results for this particular experiment: *Num_Probes* and *Segment_Mean*. There is a lot of valuable information in this file the only problem is that it needs to be organized and condensed.

```
 1 GDC_Aliquot Chromosome  Start End Num_Probes  Segment_Mean
 2 eb54eaca-ac4b-468b-a533-c8162e042c50  1 62920 16868660  8706  0.0005
 3 eb54eaca-ac4b-468b-a533-c8162e042c50  1 16877002  16934753  46  0.3682
 4 eb54eaca-ac4b-468b-a533-c8162e042c50  1 16934768  21991690  3348  0.0025
 5 eb54eaca-ac4b-468b-a533-c8162e042c50  1 21992508  21996664  2 -1.726
 6 eb54eaca-ac4b-468b-a533-c8162e042c50  1 22001786  22004046  3 -7.835
 7 eb54eaca-ac4b-468b-a533-c8162e042c50  1 22010750  22011632  2 -2.2072
 8 eb54eaca-ac4b-468b-a533-c8162e042c50  1 22015896  64961923  25865 0.0083
 9 eb54eaca-ac4b-468b-a533-c8162e042c50  1 64963532  64964291  7 -0.7467
10 eb54eaca-ac4b-468b-a533-c8162e042c50  1 64969380  72284670  4779  0.0114
```

Figure 3.4: Top 10 lines of a *txt* file containing raw data from a Gene Expression experiment.

Table 3.2 illustrates the goal of pre-processing raw data. Here we have 2 samples from different patients, each sample is a row, each column is a gene and the table is the experiment, in this case the *Gene Expression zscore*. The information is organized and only contains useful data, meaning it only has *zscore* values of each gene for each sample. It can be easily stored in a data structure. As opposed to the original data, the results are easier to understand and concentrated in one data structure instead of multiple files.

Table 3.2: Data from *Gene Expression* experiment after pre-process.

sampleId	**ACAP3_116983**	**ACTRT2_140625**	**...**	**ZXDB_158586**
TCGA-HU-A4H2-01	-1.0	-1.0		1.1309
TCGA-CG-4444-01	1.0	1.0		1.2817

The pre-processing task is extremely important going forward and is one that requires great knowledge from the user. The person that executes this task must be able to identify useful information, extract it and build a data structure with it. Based on the example, the user has to go from *txt* files to a data structure that represents Table 3.2.

3.4.2 Use Case 2: Import

Figure 3.5: Import use case. Import the data structure to a database.

The result of the pre-processing task, Section 3.4.2, is one or more data structures that are stored in volatile memory, hence once the computer shuts down all will be lost. One of the goals of this project is to show the user the data that he has in the form of statistic metrics and graphs. In order to do so, the data needs to be stored in an infrastructure, for example a database. This has multiple benefits: no data loss on computer shutdown, a database can be accessed from different computers and it is extremely efficient and easy to run queries on our data.

In order to extend the flexibility of the database, its the user responsibility to load it with the organized data. There could be an automated function to do it but it would restrain the input and thus be less versatile. However, there has to be a minimum structure level on the data.

```
1  COLLECTIONS_CSV = ["cna.csv", "methylation_hm450.csv", "rnaZscore.csv"]
2
3  # Inserts a document in the database and collection provided
4  def insert_document(database, collection, document):
5      database[collection].insert_one(document)
6
7  # Runs through the csvs to populate the database
8  def populate_database(csvs_folder, database):
9      cna_csv = csv.DictReader(open(csvs_folder + COLLECTIONS_CSV[CNA]))
10     methylation_csv = csv.DictReader(open(csvs_folder + COLLECTIONS_CSV[
   METHYLATION]))
11     rna_csv = csv.DictReader(open(csvs_folder + COLLECTIONS_CSV[RNA]))
12
13     for cna_document in cna_csv:
14         insert_document(database, COLLECTIONS_NAMES[CNA], cna_document)
15
16     for methylation_document in methylation_csv:
17         insert_document(database, COLLECTIONS_NAMES[METHYLATION],
   methylation_document)
18
19     for rna_document in rna_csv:
20         insert_document(database, COLLECTIONS_NAMES[RNA], rna_document)
```

Listing 3.1: Example of an import task.

Listing 3.1 illustrates what the implementation of an import task would be like. In our example we have a clinical trial with three experiments: *Copy Number Analysis*, *Gene Expression* and *Methylation 450*. For simplicity there is no pre-processing task needed here, the *Comma Separated Values*(CSV) files from line 1 already contain the data to push onto the database. Each database represents a clinical trial, inside a database we can have multiple experiments here called *collections* and each collection has one *document* per sample that contains the results of the experiment for that sample and the sample identification.

3.4.3 Use Case 3: View Metadata

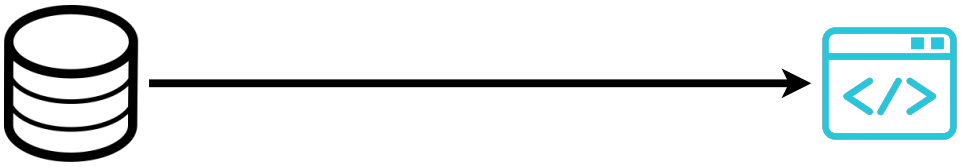


Figure 3.6: View metadata use case. Use the database information to display metadata to the user.

Once the data is stored on an infrastructure, this framework must be able to display its contents. One way of doing that is by showing some metadata to the user: number of clinical trials done, the names of the clinical trials, how many experiments were done in a specific trial and their names, the number of subjects that realized a particular test and what features were extracted. This will be invaluable information for our users because it provides an extremely convenient and hierarchical view of the data.

For our setting example the following metadata would be displayed:

- Stomach Cancer
  - Experiments:
    - *Copy Number Analysis*
      - Features: ACAP3_116983, ACTRT2_140625, ... ZXDB_158586
      - Samples: 16
    - *Gene Expression*
      - Features: ACAP3_116983, ACTRT2_140625, ... ZXDB_158586
      - Samples: 14
    - *Methylation 450*
      - Features: ACAP3_116983, ACTRT2_140625, ... ZXDB_158586
      - Samples: 16

One clinical trial: stomach cancer; Three experiments: *Copy Number Analysis*, *Gene Expression*, *Methylation 450*. The features of each test and the number of samples used. As a side note, it is not mandatory that tests include all patients of a given clinical trial as illustrated here.

3.4.4 Use Case 4: View Statistics

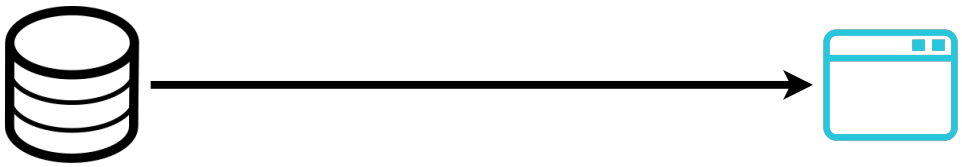


Figure 3.7: View statistics use case. Use the database information to display statistics to the user.

As a way to supplement the information displayed to the user from Section 3.4.3, graphs and statistic metrics will be used to display information about the features extracted. The statistical distribution of a feature can be easily captured in a histogram chart, if a feature only has a small number of values a pie chart would be useful to see the more common ones, maximum, minimum, mean and median of the values for a feature are frequently used.

Figure 3.8 and Table 3.3 are an example of this use case applied to the *ACAP3_116983* feature extracted from the *Gene Expression* experiment in our scenario. The values used to generate the graphs and metrics are merely illustrative.

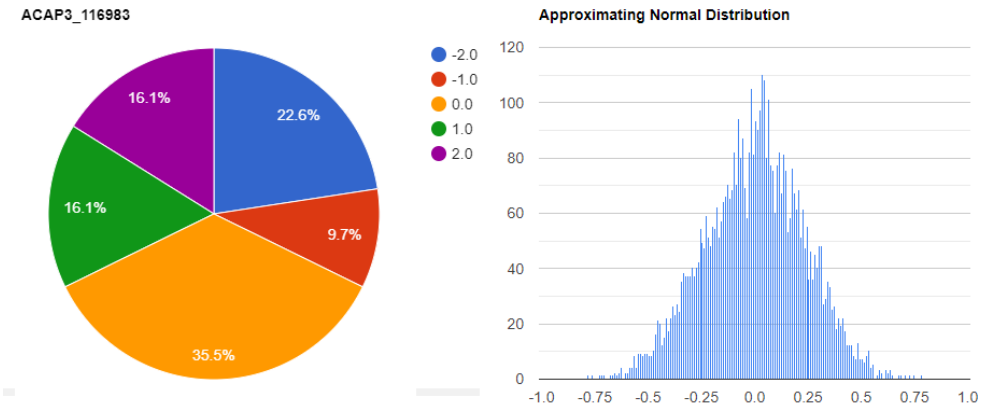


Figure 3.8: View statistics example.

Table 3.3: Example of **ACAP3_116983** maximum, minimum, mean and median values.

max	2.0
min	-2.0
mean	-0.0645
median	0.0

3.4.5 Use Case 5: Export

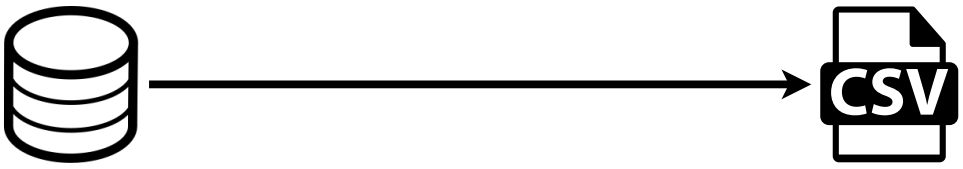


Figure 3.9: Export use case. Export a dataset to a predefined file format.

Given the tools to manipulate and view data the user needs a way to export a dataset for modeling. The framework must be able to export to different file formats and with a number of predefined settings. While exporting, a couple variances have to be taken in to account and chosen by the user: the class label, predictive models that can forsee if a new sample is healthy or sick or perhaps what condition a new sample has - A, B or C; experiments to include and how to combine them - all in the same dataset or in different datasets for stacking models; insert all features or perform some

sort of feature selection; and at last the evaluation method, divide the dataset in train and test files or in multiple files of training and testing for *K-Fold Validation*.

Given our scenario and considering that we have two classes: healthy and sick. Figure 3.10 exhibits an example for an export task. Here we have two CSV files: *dataset_train.csv* and *dataset_test.csv*. As the name indicates the first file must be used for training the model and the second one for testing it. There are only two files because the evaluation chosen is the *Hold-out* method. The training file contains 70% of the samples and the test file contains the remaining 30% using a stratified sampling, the ratio per class is equal between the two files.

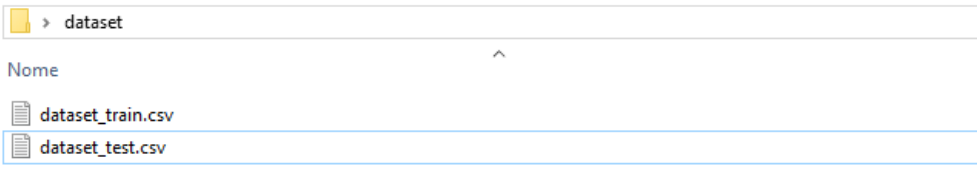


Figure 3.10: Export example.

3.4.6 Use Case 6: Process

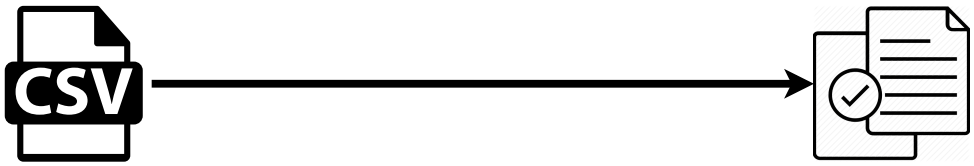


Figure 3.11: Process use case. Transform the data from a formated file into a data structure.

With the dataset created the next step is to process it before we can use it to train a model to predict the class of new samples, for instance most ML algorithms do not work if the representation given contains missing values. There are several common methods that are used, such as standardization, transformation, normalization, encoding, discretization, imputation of missing values, among others. The framework must be extendable with other frameworks that already exist and most users will be familiar with. One of the examples would be the *Pandas* framework, that easily allows the manipulation of tabular data.

Below, Listing 3.2 demonstrates how it is possible to load our example dataset from Section 3.4.5 with *Pandas*. After loading both files on lines 3 and 5, we use a function from *Pandas* called *dropna* to remove entire samples or features, *i.e.* rows or columns that only contain missing values. Then the remaining missing values are replaced by the mean of that feature with the *fillna* function. The following notation is often used in ML: X represents the features vector and y the result we want to achieve, in our case a label. As the names suggest, *X_train* and *y_train* are used to train the model and *X_test* and *y_test* to test it.

```
1 import pandas as pd
2
```

```
data_train = pd.read_csv("dataset/dataset_train.csv", index_col=0)

data_test = pd.read_csv("dataset/dataset_test.csv", index_col=0)

data_train = data_train.dropna(how='all')
data_test = data_test.dropna(how='all')

data_train = data_train.dropna(how='all', axis=1)
data_test = data_test.dropna(how='all', axis=1)

data_train = data_train.fillna(data_train.mean())
data_test = data_test.fillna(data_test.mean())

X_train = data_train.drop('CANCER', axis=1)
y_train = data_train['CANCER']

X_test = data_test.drop('CANCER', axis=1)
y_test = data_test['CANCER']
```

Listing 3.2: Example of a simple process task.

As mentioned before *Pandas* is just an example. There are many more good libraries, tools or frameworks that serve different purposes but are all useful for processing feature vectors.

3.4.7 Use Case 7: Modeling

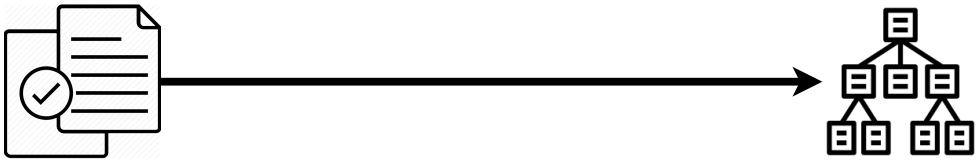


Figure 3.12: Modeling use case. Use the data structure to create a model.

With the representation created from the previous step we are able to create a model to predict the class of new samples. The framework, once again, must support the import of other ML frameworks that already implement ML algorithms that generate models, as stated in Section 3.2 it is not the purpose of this project to reinvent the wheel, we will use existing technology whenever possible and practical. For example *Scikit-learn* framework is a famous framework with a huge community and support, that easily allows the creation and training of models using popular ML algorithms.

Below, Listing 3.3 demonstrates how simple it is to create and train a classifier model using SVM and Random Forest with the *Scikit-learn* framework. *X_train* contains the features for each sample of the training dataset and *y_train* the classes that they belong to.

```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
```

```
3
4 # SVM Model
5 svm_classifier = svm.SVC()
6 svm_classifier.fit(X_train, y_train)
7
8 # Random Forest Model
9 rf_classifier = RandomForestClassifier(n_estimators=1000, max_depth=5)
10 rf_classifier.fit(X_train, y_train)
```

Listing 3.3: Example of a simple modeling task.

As stated before *Scikit-learn* is just an example. There are other libraries, tools or frameworks that provide functions to generate models. They all have advantages and disadvantages but in the end it is the users choice since our framework will allow the import of the most popular frameworks.

3.4.8 Use Case 8: Evaluate

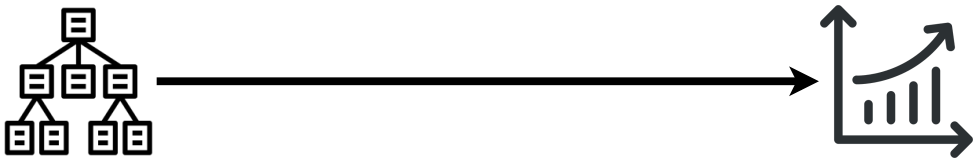


Figure 3.13: Evaluate use case. Evaluate the model created.

After creating a model we need to know how well it performs. There are several techniques used to test a model. One of the simpler ones, and the one that we use in our scenario, is to divide our dataset into two parts: training and testing. When training we only fit the model with training data, this ensures that the testing samples are unknown data to our model and that is the end goal to be able to predict the class of new samples. This exemplifies the *Hold-out* method, as the name suggests we are holding out the test samples from our model. There are other Cross Validation methods that can be used and even parameter tuning to improve the model. The last step is to generate metrics that can finally tell us how good our model really is. For classification problems we have: accuracy, lost, precision, recall, f-measures, among others.

Listing 3.4 is a simple example of an evaluation task, where we use the modela created in Section 3.4.7 to predict the classification of our testing samples and print out the accuracy of the prediction by comparing the predicted values with the real ones, lines 5 and 9.

```
1 from sklearn import metrics
2
3 # SVM Model
4 y_pred = svm_classifier.predict(X_test)
5 print("SVM model accuracy:", metrics.accuracy_score(y_test, y_pred))
6
7 # Random Forest Model
```

```
y_pred = rf_classifier.predict(X_test)
print("Random Forest model accuracy:", metrics.accuracy_score(y_test, y_pred))
```

Listing 3.4: Example of a simple evaluate task.

As said before the framework must be able to include other tools or frameworks and allow the user to leverage them in order to accomplish each task. Above we have another example with the *Scikit-learn* framework.

3.4.9 Use Case 9: Report

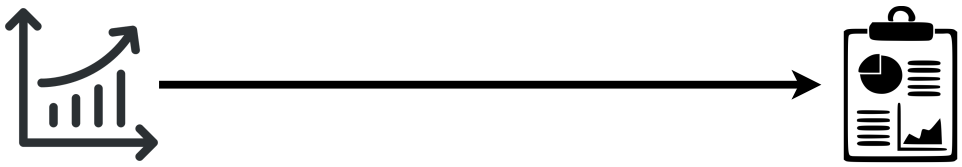


Figure 3.14: Report use case. Create a report of the whole process.

At the end of each project it is very important to report all the work done and its results. The framework must allow the user to gradually build this report while working on the project and document each step of the process. The report needs to be capable of displaying rich text, code, graphs, images, tables, formulas, among others. The user is responsible for documenting each step of the project and the report quality is directly connected to the documentation capability of the project owner.

Figure 3.15 is an example of what a report of this framework would look like. There are titles and formatted text, formulas, *Python* code and the output, graphs.

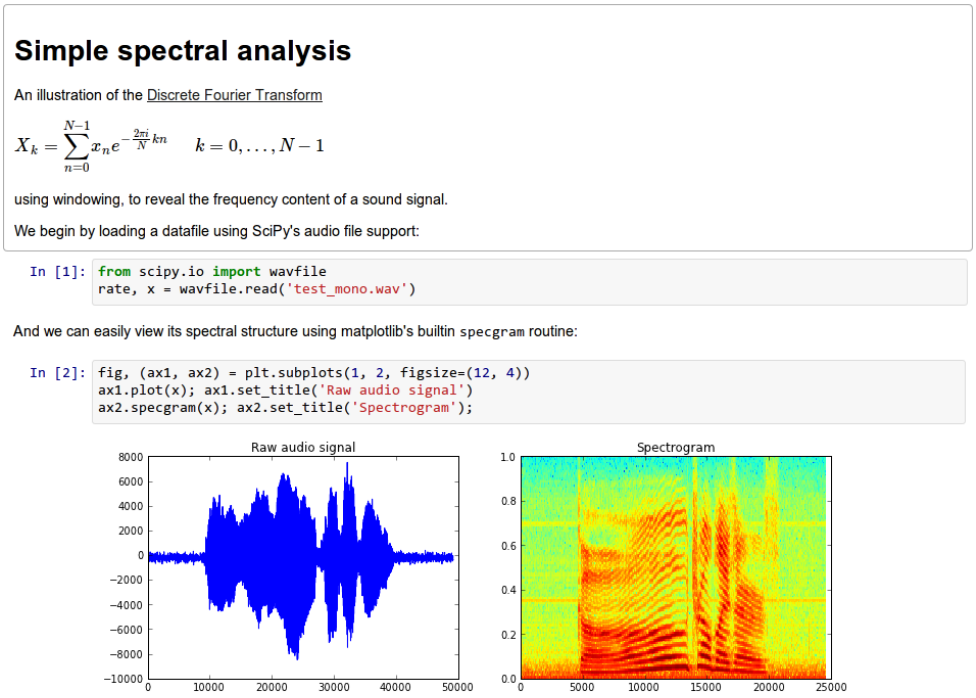


Figure 3.15: Report use case example.[1]

This file would be the user notebook, where all the computational tasks declared in Section 3.2 can be performed and documented. It is not the goal of this project to build a notebook application from scratch but instead use one that already exists.

3.5 Dissertation Questions

With the use cases above mentioned in mind we can detect a few questions about this dissertation:

1. **How can an user interface be versatile enough to allow the user to preprocess raw data and to give it a structure?** — Based on the use cases 3.4.1 and 3.4.2, it is needed an user interface that can perform calculations and manipulation on an unknown format and push that data to a structure;

2. **What is the minimum restrictions that one can apply to the data structure to be able to display the maximum variety of data?** — On the use cases 3.4.3 and 3.4.4 data must be displayed to the user, that means that it has to be structured. But structure means restriction to its form, which can be hard on diverse and complex data;

3. **What information is important to display?** — Use cases 3.4.3 and 3.4.4 demonstrate that it is a requirement to display relevant information back to the user, it is very hard to finding relevant data when you don't know the original input;

[1] https://arogozhnikov.github.io/2016/09/10/jupyter-features.html

4. **How can a user interface be powerful enough to allow the user to process, model, evaluate and export the users work?** — In 3.4.5, 3.4.6, 3.4.7 and 3.4.8 use cases it is noticeable that a user interface that allows processing, modeling, evaluation and reporting of data needs to be powerful and must not restrict the user to few choices, in other words, it needs to be open;

5. **Can the user interface from 1 and 4 be the same?** — Both need high amounts of options to do things in very different ways and both require computational tasks.

3.6 Validation

It was mentioned before that the main objective of this project is to create a ML framework to work with genomic data that is composed of existing frameworks and technologies but also new features. In addition, the framework needs provide simple access and feel like a single environment meant to ease and improve users workflow.

The validation of this work will be done in two on the steps:

The first step will consider if all the use cases mentioned in 3.4 are satisfied. These features are the requirements of this project and they are present in almost all ML projects and its phases, from raw data to the final report. All use cases are meant to be fulfilled and represent a metric to compare this project with existing similar tools in a pragmatic and objective manner.

The second step is a more subjective approach, where a ML problem will be solved using this tool and the experience of the overall feel, convenience and simpleness of the framework will be thoroughly examined.

Moreover, the results of the application of this tool will be submitted in related papers or articles for conferences in the area.

In the end the perspective that both measures will provide can showcase the benefits, convenience and capabilities of this project.

3.7 Expected Contributions

Creating a framework that can solve all the questions of Chapter 3.5 under the same environment for a easy to use and workflow based tool. In here we present the answers to those questions based on what our tool is expected to became:

1. **Versatile user interface, enough to allow the user to preprocess raw data and to give it a structure:** — This would be a perfect job for an interpreter for a programming language. A *Jupyter* notebook gives the user that kind of versatility, providing the user with a built-in *Python* interpreter to solve these tasks;

2. **Minimum restrictions that one can apply to the data structure to be able to display the maximum variety of data:** — The solution here is to have each clinical trial in its own *Mongo* database and each of the experiments be a collection of that database;

3. **Important information to display:** — Has to be generalized to some metrics specific to the context (*eg.* total and unique subjects or samples) and to statistics (*eg.* mean, median, maximum, minimum, distribution);

4. **Powerful enough user interface to allow the user to process, model, evaluate and export the users work:** — This is a well known use case for a *Jupyter* notebook. With the use of *Python* and *Markdown* one can process and model data in a report like document;

5. **User interface from 1 and 4 combined:** — *Jupyter* user interface has the ability to create multiple notebooks, meaning that for all the computational tasks this project will leverage the resources of *Jupyter* notebooks.

3.8 Conclusions

The objective of this work is to provide an unified framework for users that work with genomic data that can handle all the standard tasks in a ML project.

The framework will let the user transform raw data into a organized structured set; import that data to a database that will serve as the source to statistic metrics and important information to be displayed and to allow the export of datasets with predefined settings; Load a dataset, perform feature selection and feature engineering, create models and evaluate them and in the end have a document that can be used as an initial report.

The full purpose of this project is to be a mixture of existing technologies or tools and new features that are under the same environment providing an easier more convenient way to the typical ML workflow under the scope of genomic data.

Chapter 4

Implementation

This Chapter will tackle the problems found in Chapter 3 with a system that implements all the requirements mentioned in Section 3.4.

Initially there will be an overview of technologies used and the project different parts from a high level point of view. In this section is an initial statement about architecture, technologies, components and how they relate.

Next, the architecture of the system is brought to its gears and bearings to a comprehensive description that demonstrates how the tasks mentioned in section 1.3 can be completed using this framework, how everything is connected and which components exist and their responsibilities.

The last section gives an outline of an example that can be used in order to validate and demonstrate the framework.

4.1 Technologies Used

4.1.1 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line

at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

4.1.2 Jupyter Notebook

Jupyter Notebook[1] documents or just *notebooks* are documents produced by the Jupyter Notebook App, which contain both computer code (*e.g.* python) and rich text elements (*e.g.* paragraph, equations, figures, links, among others). Notebook documents are both human-readable documents containing the analysis description and the results (*e.g.* figures, tables and graphs) as well as executable documents which can be run to perform data analysis.

The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet. In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a "Dashboard" (Notebook Dashboard), a "control panel" showing local files and allowing to open notebook documents or shutting down their kernels.

A notebook kernel is a "computational engine" that executes the code contained in a Notebook document. The ipython kernel, referenced in this guide, executes python code. Kernels for many other languages exist (official kernels). When you open a Notebook document, the associated kernel is automatically launched. When the notebook is executed (either cell-by-cell or with menu Cell -> Run All), the kernel performs the computation and produces the results. Depending on the type of computations, the kernel may consume significant CPU and RAM. RAM is not released until the kernel is shut-down.

The Notebook Dashboard is the component which is shown first when you launch Jupyter Notebook App. The Notebook Dashboard is mainly used to open notebook documents, and to manage the running kernels (visualize and shutdown). The Notebook Dashboard has other features similar to a file manager, namely navigating folders and renaming/deleting files.

4.1.3 Numpy

NumPy[2] is a Python package which stands for *Numerical Python*. It is the core library for scientific computing, which contains a powerful n-dimensional array object, provide tools for integrating C, C++ etc. It is also useful in linear algebra, random number capability etc. NumPy array can also be used as an efficient multi-dimensional container for generic data.

NumPy Array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize numpy arrays from nested Python lists and access it elements in order to perform numpy operations.

[1] https://jupyter.org/
[2] https://www.numpy.org/

4.1.4 Pandas

Pandas[3] is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with *relational* or *labeled* data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R's data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets

[3] https://pandas.pydata.org

- Intuitive merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Hierarchical labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format
- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging, etc.

Many of these principles are here to address the shortcomings frequently experienced using other languages/scientific research environments. For data scientists, working with data is typically divided into multiple stages: munging and cleaning data, analyzing / modeling it, then organizing the results of the analysis into a form suitable for plotting or tabular display. pandas is the ideal tool for all of these tasks.

Some other remarks:

- pandas is fast. Many of the low-level algorithmic bits have been extensively tweaked in Cython code. However, as with anything else generalization usually sacrifices performance.
- pandas is a dependency of statsmodels, making it an important part of the statistical computing ecosystem in Python.
- pandas has been used extensively in production applications.

4.1.5 Scikit-learn

Scikit-learn[4] is a free software machine learning library for the Python programming language.

Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007. Later Matthieu Brucher joined the project and started to use it as apart of his thesis work. In 2010 INRIA got involved and the first public release (v0.1 beta) was published in late January 2010.

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

- NumPy: Base n-dimensional array package
- SciPy: Fundamental library for scientific computing

[4] https://scikit-learn.org

- Matplotlib: Comprehensive 2D/3D plotting
- IPython: Enhanced interactive console
- Sympy: Symbolic mathematics
- Pandas: Data structures and analysis

Extensions or modules for SciPy care conventionally named SciKits. As such, the module provides learning algorithms and is named scikit-learn.

The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as easy of use, code quality, collaboration, documentation and performance.

Although the interface is Python, c-libraries are leverage for performance such as numpy for arrays and matrix operations, LAPACK, LibSVM and the careful use of cython.

4.1.6 Conda

Package, dependency and environment management for any language—Python, R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, FORTRAN

Conda[5] is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on a local computer. It was created for Python programs, but it can package and distribute software for any language.

Conda as a package manager allows the user to find and install packages. With just a few commands, it can set up a totally separate environment to run different version of Python, while continuing to preserve the system Python in the normal environment.

In its default configuration, conda can install and manage thousand of packages[6] at are continuously built, reviewed and maintained by Anaconda[7].

4.1.7 Django

Django[8] is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

[5] https://github.com/conda/conda
[6] repo.continuum.io
[7] https://www.anaconda.com/
[8] https://www.djangoproject.com/

Django was designed to help developers take applications from concept to completion as quickly as possible. It includes dozens of extras that can be used to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks right out of the box.

It takes security seriously and helps developers avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords.

Some of the busiest sites on the planet use Django's ability to quickly and flexibly scale to meet the heaviest traffic demands. Companies, organizations and governments have used Django to build all sorts of applications — from content management systems to social networks to scientific computing platforms.

4.1.8 Docker

Docker[9] container technology was launched in 2013 as an open source Docker Engine.

It leveraged existing computing concepts around containers and specifically in the Linux world, primitives known as cgroups and namespaces. Docker's technology is unique because it focuses on the requirements of developers and systems operators to separate application dependencies from infrastructure.

Success in the Linux world drove a partnership with Microsoft that brought Docker containers and its functionality to Windows Server, sometimes referred to as Docker Windows containers.

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers - images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

Docker containers that run on Docker Engine:

- **Standard**: Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight**: Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure**: Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

[9] https://www.docker.com/

Technology available from Docker and its open source project, Moby has been leveraged by all major data center vendors and cloud providers. Many of these providers are using Docker for their container-native Infrastructure as a Software offerings. Additionally, the leading open source serverless frameworks utilize Docker container technology.

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than Virtual Machines (container images are typically tens of MBs in size), can handle more applications and require fewer Virtual Machines and Operating Systems.

4.1.9 Docker Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, a YAML file is used to configure an application's services. Then, with a single command, create and start all the services from the configuration file.

Compose works in all environments: production, staging, development, testing, as well as continuous integration workflows.

Using Compose is basically a three-step process:

1. Define the application environment with a Dockerfile so it can be reproduced anywhere.

2. Define the services that make up the application in docker-compose.yml so they can be run together in an isolated environment.

3. Run docker-compose up and Compose starts and runs the entire application.

A docker-compose.yml file looks like this:

```
version: '3'

services:
  datascience-notebook:
      image: amaksimov/python_data_science:anaconda
      volumes:
        - ./notebooks:/notebooks
      ports:
        - 8888:8888
```

Listing 4.1: Example of a docker-compose.yml file

In the above example we have a Compose file version 3, that only has one service named datascience-notenook. The services uses the *amaksimov/python_data_science:anaconda* image, maps the folder notebooks and the port 8888 between the host and the container.

4.2 Overview

With the goals mentioned in Section 3.2 in mind and the fact that it is not one of those goals to implement already available features, a set of existing frameworks were used to build the system.

Jupyter notebooks combine computer code with rich text elements that allow the user to have a interactive document that has runnable parts which can generate output. In Figure 4.1 is a screen shot of a *Jupyter* notebook, the first part of the document is a block of rich text written with *Markdown*, the second block is *Python* code and after running this code we get the output in the bottom of the figure.

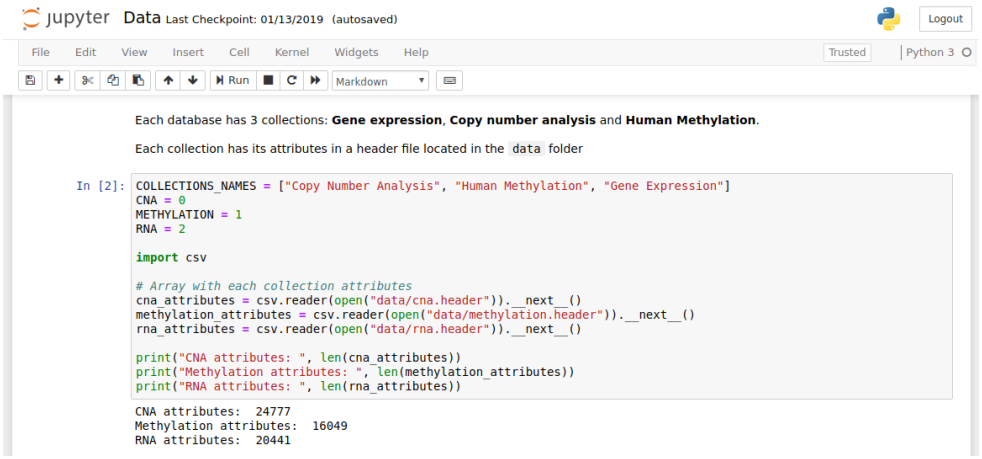


Figure 4.1: Jupyter notebook example.

These *Jupyter* notebooks will be the user interface for the computational tasks declared in Section 3.2. They allow a very interactive user interface to run computer code while annotating your work and all this within a browser.

Python is an interpreted, high-level computer language that is very beginner friendly and extremely readable. In addition, it has a huge support in libraries for ML related tasks. For those reasons, *Python* was the language of choice to develop this framework and to do its demonstration.

Mongo databases store the information as documents(Figure 4.2) the same way that *Python* dictionaries behave. So one can easily see the convenience of using a *Mongo* database when working with *Python*.

Figure 4.2: Mongo document example.

For the display tasks declared in Section 3.2 a web application was the solution found, since it also runs on a browser. As the use of *Python* was already an established decision, a *Python* framework to create this web application extended the concepts of simplicity and convenience that we aim to bring with this solution. The chosen one was *Django*, a framework to develop websites using the *Model-View-Controller* architecture.

The final goal is to have all this technologies put together in a fashion that allows the user to jump between them and use them as part of a very well established workflow.

A *Jupyter* notebook is created to preprocess the raw data into a structured database, then that processed data is pushed onto a *Mongo* database. The *Django* web application retrieves metadata and statistic measures from the *Mongo* database and displays it, the user can also use the web application to extract a dataset by passing of some predefined settings to the *app*. A new notebook can be created work with that dataset: creating models, evaluating different models and use the notebook as a report.

Below, Figure 4.3 represents an high level view of the use of the technologies above mentioned in our architecture. The system runs inside a unique host, the yellow rectangle is our framework, with two high level processes and its respective tasks as stated in Section 3.2.

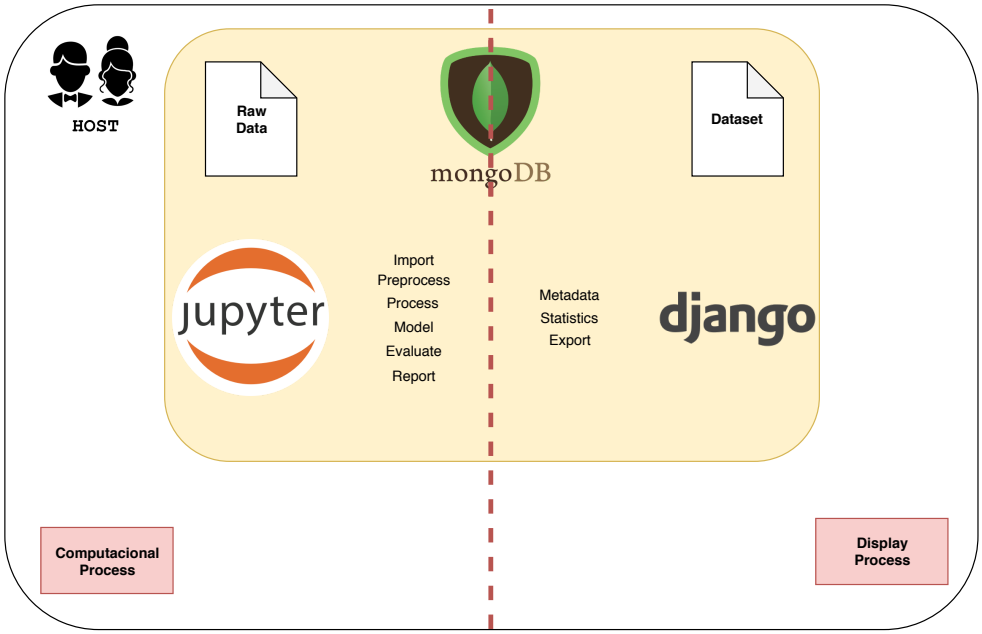


Figure 4.3: Solution proposed with the technologies chosen.

4.3 Solution

The solution is composed of three key components: *Jupyter* notebook application, *Mongo* database and *Django* web application.

In order to encapsulate and abstract the dependencies from the host each component will be running on a *Docker* container. *Docker* is a software that allows the virtualization of an entire operating system in what is called *containers*. Each container can have different applications, libraries, kernels, configurations, among others. This is very useful in this project because managing software dependencies from the three components in a single operating system is extremely hard and that is not what this project is about, simplicity here is a key factor.

To manage the multiple containers we used an application called *Docker Compose*. *Docker Compose* allows the user to define multiple *Docker* containers in a single file, this file also contains all the configurations for each container. Having a single file where a user can manage each component of the framework and configure them without the hassle of dependency management.

The idea is to have the three components in individual *Docker* containers that communicate between each other either by the local network interface or by exchanging files using the host file system.

The *Django* web application and *Mongo* database are not out-of-shelf solutions. The web application will need to be built from scratch to meet our requirements and the database needs to be our warehouse to the data that we supply. Our *Django* web application was code named *Gene Miner*.

Beneath, Figure 4.4 represents the view of all the components of the project architecture as well as the workflow of the framework. Data flow is represented by the arrows, starting in raw data. Each one of the boxes with the *Docker* symbol are containers.

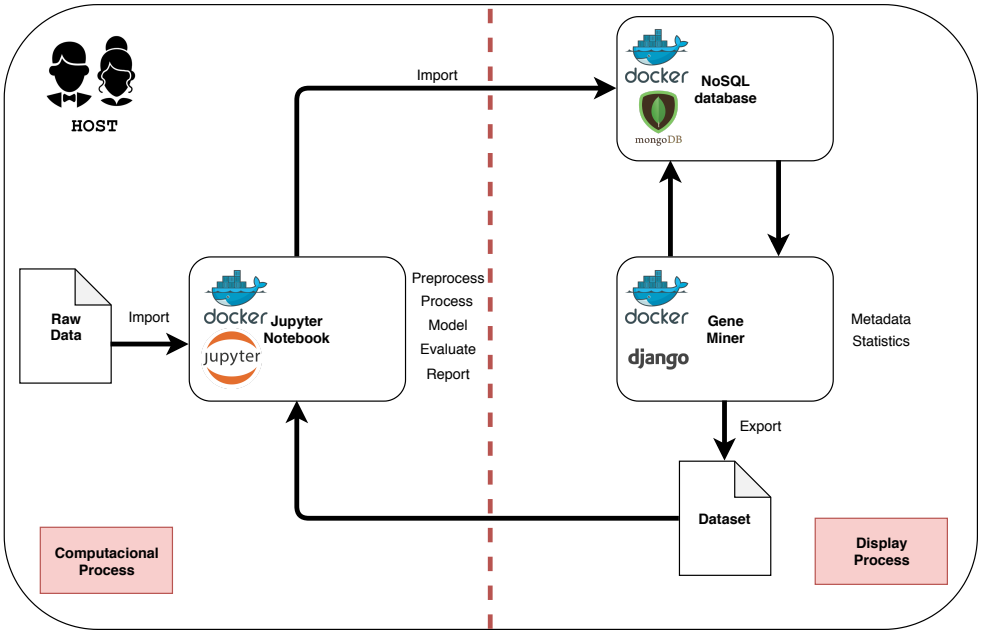


Figure 4.4: Architecture of the proposal with workflow.

4.4 Components

Within this section each individual component of the project is dissected. To be in sync with the workflow and the use cases declared in Section 3.4, the *Jupyter* container is divided into two different sections, one for the processing of the raw data and import to the *Mongo* database and the other for the processing, modeling, evaluating and reporting with the dataset.

4.4.1 Jupyter Container: Preprocess and Import tasks

It is through the use of a *Jupyter* notebook, the *Python* programming language and some libraries that the user has access to a programmable and annotated platform to transform raw unorganized data to structured information. We use a *Jupyter* notebook as the user interface at this stage because it allows the user to have extreme flexibility to address the unformatted data with an auto explanatory approach useful to the user to follow their work. Another advantage of the *Jupyter* notebook is that its a web application, it does not require nothing more than a web browser to display and work with notebooks, from a front-end perspective.

Back-end however is another issue and that is were docker containers come in handy. Instead of having the *Jupyter* application running on the host, we instead launch a *Docker* container that exposes the port 8888 on the host operating system with the web application and has all the required software for the *Jupyter* application but also project specific dependencies to handle data and push it to a database. This avoids the cumbersome task of installing every single dependency, from *Python* itself to every library that will be required to accomplish the tasks. Doing it this way, the host system remains clean, no footprint of installed dependencies that sometimes are only used in a single project.

Docker virtualization makes it possible to run the container in any system that a *Docker daemon* was developed for. This includes the three major operating systems: Microsoft Windows, Apple's MacOS and any Linux based system. The framework will work out of the box in any system that has a *Docker daemon* running. Listing 4.2 is the definition of the *Jupyter* application container. Based on the *amaksimov/python_data_science:anaconda* docker image, it exposes the *Jupyter* application to the host on the port 8888.

```
version: '3'

services:
  datascience-notebook:
      image: amaksimov/python_data_science:anaconda
      volumes:
        - ./notebooks:/notebooks
      ports:
        - 8888:8888
```

Listing 4.2: Definition of the *Jupyter* application container.

This container is built on top of *Ubuntu 16.04* and uses *Anaconda* open source distribution of *Python* language that contains over 1400 popular data-science packages. This is the list of all pre-installed packages:

- NumPy;
- Sklearn;
- Matplotlib;
- Seaborn;
- pyyaml;
- h5py;
- Jupyter;
- Tensorflow;
- Keras;
- OpenCV 3.

This implementation is very versatile, if the user needs a new package as long as it is in *Anaconda* all that is needed is to run a new cell in a notebook containing this: *!conda install name-of-the-package* (eg. *!conda install scipy*).

This is one of the examples of using already available technology and build a framework to automate processes connected to the work in scope. There is no point on developing a similar feature from scratch if one already exists, has the support of a community and is well matured.

4.4.2 Mongo Database

Mongo databases are non relational document based databases. A database inside *Mongo* is composed by *collections* and the *collections* contain *documents*. *Documents* are *JSON* like objects, which are very similar to *Python dictionaries*. For that reason, a *Mongo* database was the logical choice to hold our data after the pre-processing stage and to serve as the back-end storage of our web application. As a consequence of having the database serving not only as the infrastructure to store our data but also the back-end storage of our web application is that after the insertion of the data it also has to compute and store metadata and computed statistic measures to be displayed later.

Below, Listing 4.3 has the definition of the *Mongo* database container. It is based on the *mongo* image and exposes *Mongo* default port in the host environment, port 27017.

```
version: '3'

services:
  mongodb:
    image: mongo:latest
    environment:
      - MONGO_DATA_DIR=/data/db
      - MONGO_LOG_DIR=/dev/null
    ports:
      - 27017:27017
    volumes:
      - ./data/db:/data/db
    command: mongod --smallfiles --logpath=/dev/null # --quiet
```

Listing 4.3: Definition of the *Mongo* database container.

The only constrain applied to the data structure that needs to be respected in order for the entire system to work, it has to do with the way that data is imported to *Mongo*. Applyed to the study of genomic data, for each clinical trial that a geneticist is doing a database within *Mongo* is created. Each experiment from that trial in a clinical trial is a collection in that database that contains all the samples tested.

4.4.3 GeneMiner Web Application

A *Django* web application, named *GeneMiner*, was designed from scratch to display the metadata and statistic measures from the data inside the *Mongo* database and to be able to export datasets from that data. The export task has different predefined settings, such as: clinical trials to include, experiments to take into account, divide the datasets by experiment for stacking models later, cross validation method to be used and its parameters and the format to extract the file to, CSV or ARFF.

Listing 4.4 defines the docker container for the *Django* web application. It uses a custom image named *webapp* and exposes the web application on port 8000 in the host environment.

```
version: '3'

services:
  django:
    build: ./webapp
    ports:
      - 8000:8000
    volumes:
      - ./webapp:/webapp
    command: python3 manage.py runserver 0.0.0.0:8000
```

Listing 4.4: Definition of the *Django* web application container.

The GeneMiner applications consists of two main sections, one for browsing data statistics and the other to export a dataset. Each section with its own set of menus and features shown in Tables 4.1 and 4.2.

Table 4.1: Statistics section menus.

Menu	**Features**
Clinical Trials	List of existing clinical trials For each trial a list of the experiments conducted
Clinical Trial	List of the experiments conducted in the trial selected Number of samples respective to each individual experiment Number of attributes/features each individual experiment has
Experiment	List all attributes that belong to the experiment selected A search box to find a particular attribute
Attribute	Statistic metrics of the attribute selected across all samples Graphs about the selected attribute values

Table 4.2: Export section menus.

Menu	**Features**
Clinical Trials	Select the clinical trials that will be on the dataset
Export	Export the dataset based on these options: Experiments to include Validation technique that will be used With or without feature selection Whether or not to use stacking

4.4.4 Jupyter Container: ML tasks

The container used in Section 4.4.1, represented in Listing 4.2 is also used in this component. To accomplish this we leverage one of the features of the *Jupyter Application*, which allows the user to create multiple notebooks. Thus, the user can create one notebook to handle the pre-processing of raw, unorganized data and import it to a database and another notebook to read a dataset created by our *Django Web Application* and process, model, evaluate and report the results. As mentioned before it is not the goal of this project to reinvent existing technologies but rather to take advantages of existing application whenever possible, this is a good example of that premise.

The user is responsible for the notebook contents, this applications only provides the means to reach the end goal. That goal being, a step by step report from dataset to results and evaluation. The structure of the notebook, frameworks chosen, ML algorithms that will be used to generate models, evaluation of those models and presentation of the full report depend solely on the user skills and expertise.

4.5 Assembling the Framework

By assembling all the components for this solution it is possible to have a framework that implements the workflow that handles genomic based cancer studies using Machine Learning algorithms as described in Section 3.2.

Listing 4.5 presents a sample *docker-compose.yml* file that allows the *Jupyter Application*, *Mongo* database and *GeneMiner Application* to be deployed in a host system that has a running Docker daemon with a single bash command. The three components share the same network and filesystem in order to communicate and exchange files.

```
version: '3'

services:
  django:
    build: ./webapp
    ports:
      - 8000:8000
    volumes:
      - ./webapp:/webapp
    command: python3 manage.py runserver 0.0.0.0:8000
  mongodb:
    image: mongo:latest
    environment:
      - MONGO_DATA_DIR=/data/db
      - MONGO_LOG_DIR=/dev/null
    ports:
      - 27017:27017
    volumes:
      - ./data/db:/data/db
    command: mongod --smallfiles --logpath=/dev/null # --quiet
  datascience-notebook:
      image: amaksimov/python_data_science:anaconda
      volumes:
        - ./notebooks:/notebooks
      ports:
        - 8888:8888
```

Listing 4.5: Docker Compose yaml file for the framework.

Docker Compose is a tool able to define and run multiple Docker applications that may be networked or depend on each other. *Docker*, as explained before, is a software that allows the virtualization of an entire operating system in what is called *containers*. Each container can have different applications, libraries, kernels, configurations, among others.

4.6 Conclusions

The framework takes advantages of technologies and techniques that already exist and to provide geneticists with a feature rich tool to study cancer related work with the application of ML algorithms. From a software point of view the framework is composed by three components with the boundaries well exposed, it could be easily transformed to a closed system to convey the environment like experience, *i.e.* give the user the impression of a single component.

It becomes important to understand exactly how well the system works and how it delivers its proposed features. Chapter 5 describes the methods used to evaluate the solution and presents the results testing the framework.

Chapter 5

Validation

This chapter demonstrates how the solution implemented for this project accomplishes the features proposed in Chapter 3.

The first section will explain how the use cases mentioned in Section 3.4 are satisfied. These features are the requirements of this project and represent the workflow of the platform from raw data to the final report. All use cases were fulfilled and represent a metric to compare this project with existing similar tools in a pragmatic and objective manner.

The second section takes a more subjective approach, where a practical example is solved using this tool and the experience of the overall feel, convenience and simpleness of the framework are examined.

In the end the perspective that both measures provide can showcase the benefits, convenience and capabilities of this project.

5.1 Use Cases

The first section will explain how the use cases mentioned in Section 3.4 are satisfied. These features are the requirements of this project and represent the workflow of the platform from raw data to the final report. All use cases were fulfilled and represent a metric to compare this project with existing similar tools in a pragmatic and objective manner.

This section explains how the features of the framework referenced in Section 3.4 are satisfied by its three components and how the user can make use of them. Although some of the features are intrinsic to the underlying technologies the overall combination of existing and new technologies into a framework is what adds value to the presented solution.

5.1.1 Use Case 1: Pre-process

Figure 5.1: Preprocessing use case. Transform raw data into a data structure using Jupyter.

The framework is capable to pre-process raw data leveraging the capabilities of the *Jupyter* notebooks, the *Python* language and its packages it is possible to import raw data that contains features, in the form of results, extracted from experiments conducted during the clinical trial and transform that data into a organized and well defined data structure for later use.

It is an absolute requirement that the user must possess complete knowledge over the raw data information, otherwise he will not be able to retrieve and handle the necessary information required going forward.

5.1.2 Use Case 2: Import

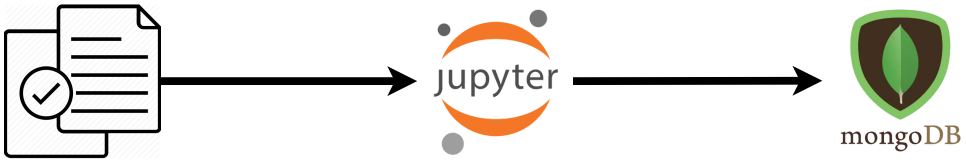


Figure 5.2: Import use case. With Jupyter import the data structure to a Mongo database.

The result of the pre-processing task, is one or more data structures that need to be stored in an database. We continue to use the *Jupyter* notebook from the previous section in order to push the data to the *MongoDB* component.

Figures 5.3 and 5.4 are excerpts of a *Jupyter* notebook with an import example of three clinical trials (skin,stomach and thyroid cancer) with three experiments each (Copy Number Analysis, Human Methylation and Gene Expression). For simplicity, the data for each experiment is already preprocessed and it is loaded from *CSV* files.

Data

Create 3 databases for each type of cancer. The different types in study are **skin**, **stomach** and **thyroid** cancers.

In [1]:
```
import pymongo

DATABASE_NAMES = ["skin_cancer_db", "stomach_cancer_db", "thyroid_cancer_db"]
SKIN = 0
STOMACH = 1
THYROID = 2

mongo_client = pymongo.MongoClient("mongodb://localhost:27017/")

mongo_client.drop_database(DATABASE_NAMES[SKIN])
mongo_client.drop_database(DATABASE_NAMES[STOMACH])
mongo_client.drop_database(DATABASE_NAMES[THYROID])

skin_cancer_db = mongo_client[DATABASE_NAMES[SKIN]]
stomach_cancer_db = mongo_client[DATABASE_NAMES[STOMACH]]
thyroid_cancer_db = mongo_client[DATABASE_NAMES[THYROID]]

databases = [skin_cancer_db, stomach_cancer_db, thyroid_cancer_db]

print("MongoDB:", mongo_client.server_info)
```

```
MongoDB: <bound method MongoClient.server_info of MongoClient(host=['localhost:27017'], documen
t_class=dict, tz_aware=False, connect=True)>
```

Each database has 3 collections: **Gene expression**, **Copy number analysis** and **Human Methylation**.

Each collection has its attributes in a header file located in the `data` folder

In [2]:
```
COLLECTIONS_NAMES = ["Copy Number Analysis", "Human Methylation", "Gene Expression"]
CNA = 0
METHYLATION = 1
RNA = 2
```

Figure 5.3: Import use case: connect to *Mongo* and create the databases.

The data for each database is inside 3 csv files (one per collection) inside `skin`, `stomach` and `thyroid` folders under the main `data` folder. Extract the data for each database and for each database's collection and insert it into MongoDB.

In [3]:
```
COLLECTIONS_CSV = ["cna.csv", "methylation_hm450.csv", "rnaZscore.csv"]

# Inserts a document in the database and collection provided
def insert_document(database, collection, document):
    database[collection].insert_one(document)

# Runs through the csvs to populate the database
def populate_database(csvs_folder, database):
    cna_csv = csv.DictReader(open(csvs_folder + COLLECTIONS_CSV[CNA]))
    methylation_csv = csv.DictReader(open(csvs_folder + COLLECTIONS_CSV[METHYLATION]))
    rna_csv = csv.DictReader(open(csvs_folder + COLLECTIONS_CSV[RNA]))

    for cna_document in cna_csv:
        insert_document(database, COLLECTIONS_NAMES[CNA], cna_document)

    print("\tNumber of CNA records:", cna_csv.line_num-1)

    for methylation_document in methylation_csv:
        insert_document(database, COLLECTIONS_NAMES[METHYLATION], methylation_document)

    print("\tNumber of Methylation records:", methylation_csv.line_num-1)

    for rna_document in rna_csv:
        insert_document(database, COLLECTIONS_NAMES[RNA], rna_document)

    print("\tNumber of RNA records:", rna_csv.line_num-1)


# skin_cancer_db
print("Skin Cancer Database:")
populate_database("data/skin/", skin_cancer_db)

# stomach_cancer_db
print("Stomach Cancer Database:")
populate_database("data/stomach/", stomach_cancer_db)

# thyroid_cancer_db
print("Thyroid Cancer Database:")
populate_database("data/thyroid/", thyroid_cancer_db)
```

```
Skin Cancer Database:
        Number of CNA records: 367
        Number of Methylation records: 473
        Number of RNA records: 472
Stomach Cancer Database:
        Number of CNA records: 441
        Number of Methylation records: 397
        Number of RNA records: 415
Thyroid Cancer Database:
        Number of CNA records: 499
        Number of Methylation records: 567
        Number of RNA records: 509
```

Figure 5.4: Import use case: populate the databases with data.

5.1.3 Use Case 3: View Metadata

Figure 5.5: View metadata use case. Use the Mongo database information to display metadata to the user in a browser.

Once the data is stored in *Mongo*, this framework is able to display its contents using the *GeneMiner* application. Some metadata can be browser by the user: number of clinical trials done, the names of the clinical trials, how many experiments were done in a specific trial and their names,

the number of subjects that realized a particular test and what features they contain. This is invaluable information for a researcher because it provides an extremely convenient and hierarchical view of the data.

Below, Figures 5.6, 5.7 and 5.8 showcase the hierarchical view of the metadata in the *GeneMiner* application.

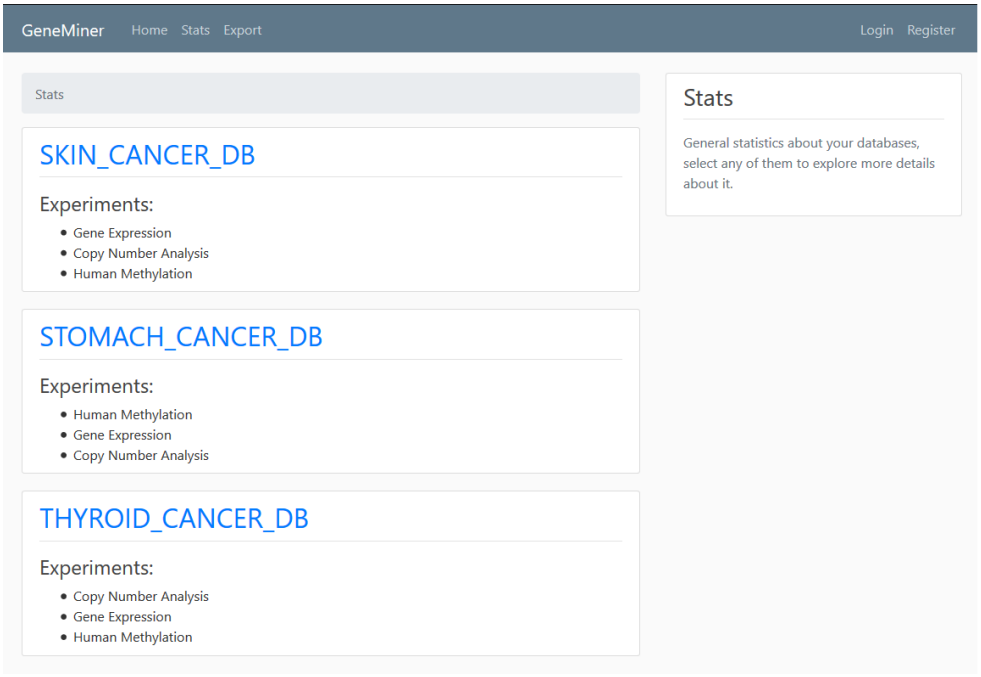


Figure 5.6: General statistics about the clinical trials.

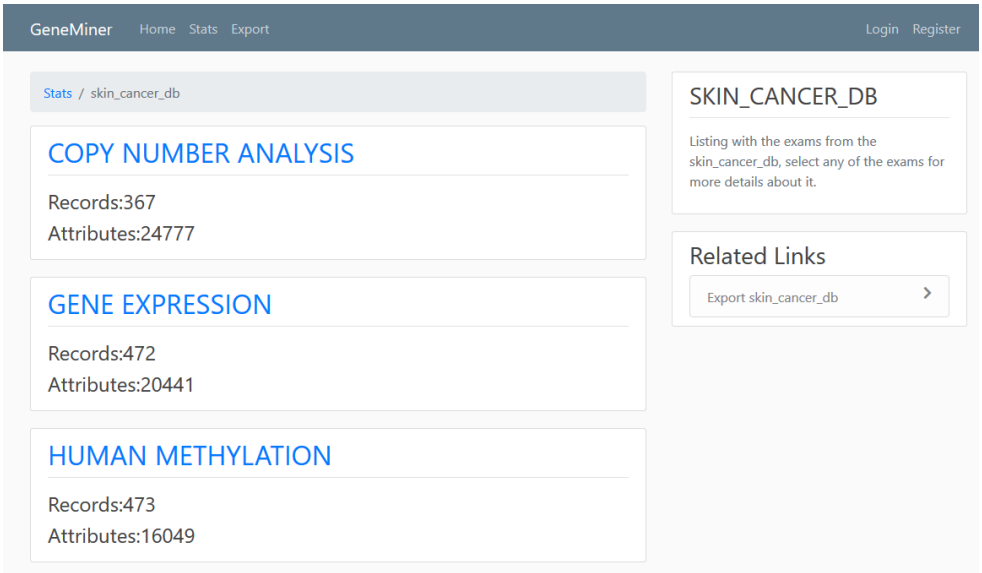


Figure 5.7: Listing with the exams from the skin cancer trial.

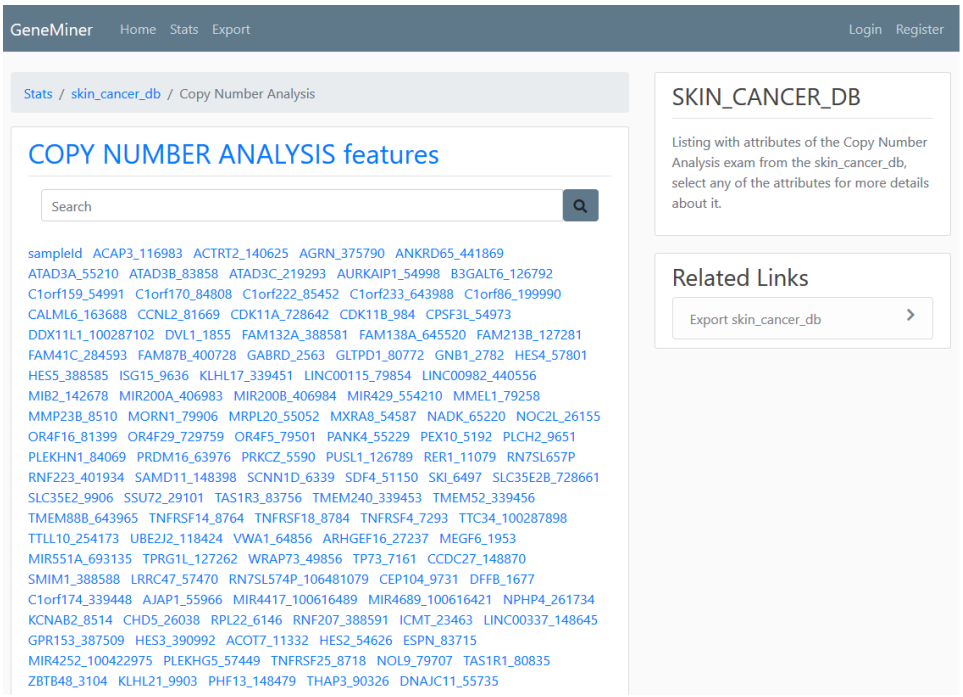


Figure 5.8: Searchable listing with attributes of the Copy Number Analysis exam from the skin cancer trial.

5.1.4 Use Case 4: View Statistics

Figure 5.9: View statistics use case. Use the Mongo database information to display statistics to the user in a browser.

As a way to supplement the information displayed to the user from from the previous section, graphs and statistic metrics are used to display information about the features extracted. These metrics are computed in the *Mongo* component after the import stage. The statistical distribution of a feature can be easily captured in a histogram chart, if a feature only has a small number of values a pie chart it will be generated to see the more common ones, maximum, minimum, mean and median of the values for a feature also are presented as Figure 5.10.

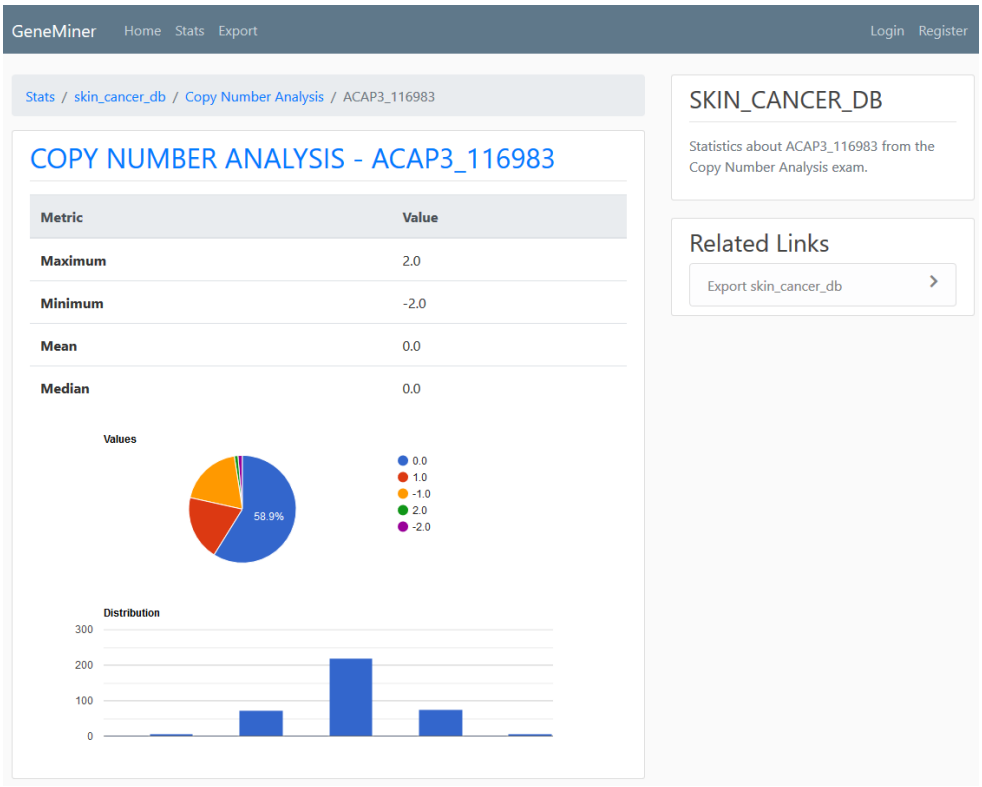


Figure 5.10: Statistics about *ACAP3_116983* from the Copy Number Analysis exam from the skin cancer trial.

5.1.5 Use Case 5: Export

Figure 5.11: Export use case. Export a dataset to a CSV or ARFF file format in the django web application.

Given the tools to manipulate and view data the user needs a way to export a dataset for modeling. The *GeneMiner* application from the framework is able to export to different file formats and with a number of predefined settings. While exporting, the researcher can choose from different settings: clinical trials to include, if only one is selected the study will be based on healthy versus cancer samples, but if the user selects more than one the study is about cancer versus cancer samples; experiments to include and how to combine them - all in the same dataset or in different datasets for stacking models; insert all features or perform some sort of feature selection; and at last the evaluation method, divide the dataset in train and test files or in multiple files of training

and testing for *K-Fold Validation*. Figures 5.12 and 5.13 illustrate the process using the *GeneMiner* application.

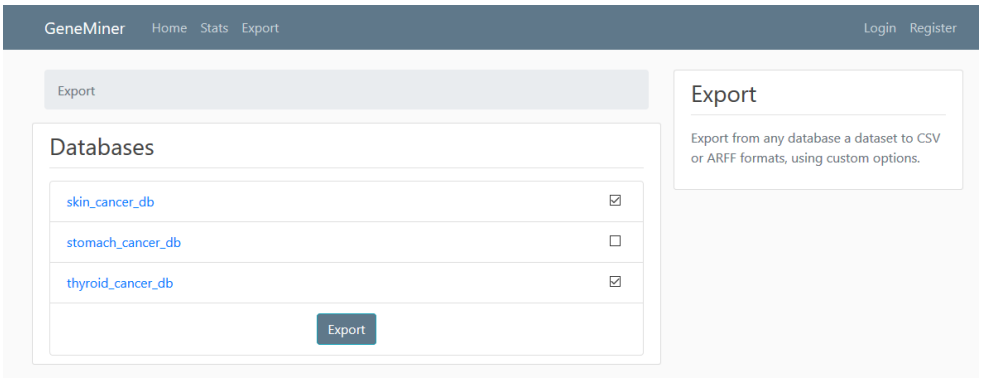


Figure 5.12: Select one or more clinical trials for the creation of the dataset.

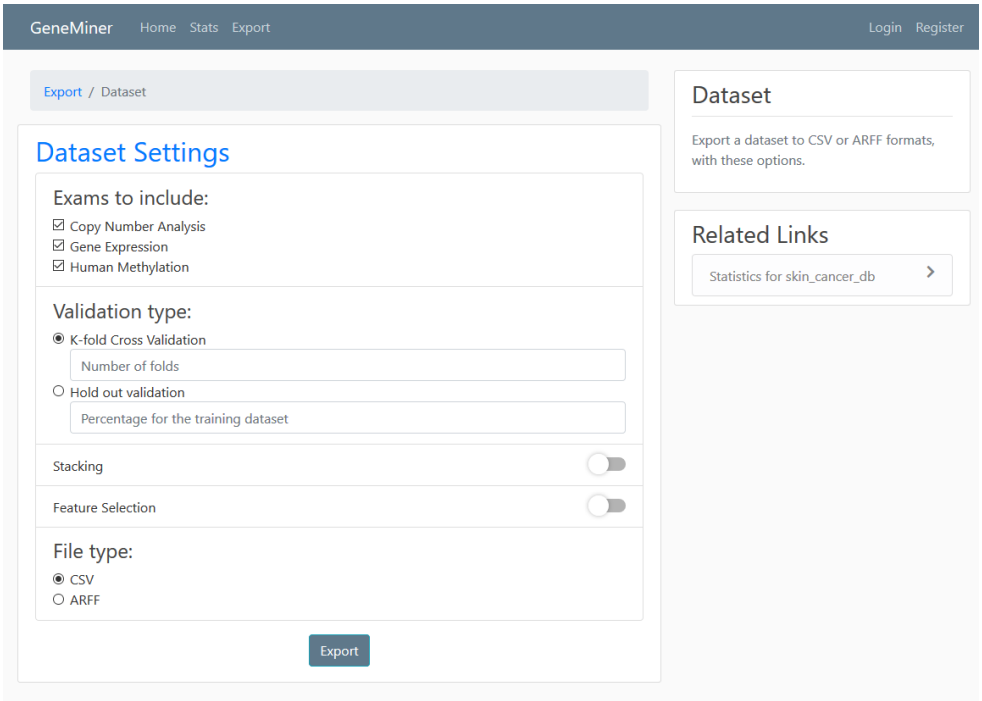


Figure 5.13: Select settings to export the dataset.

5.1.6 Use Case 6: Process

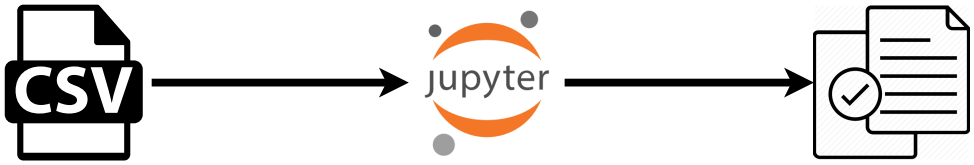


Figure 5.14: Process use case.

With the dataset created the next step is to allow a researcher to process it in order to to train a model to predict the class of new samples. Most ML algorithms do not work if the representation given contains missing values, this and another issues are addressed in this stage. There are several common methods that are used, such as standardization, transformation, normalization, encoding, discretization, imputation of missing values, among others. Figure 5.15 is an excerpt of a *Jupyter* notebook to handle missing values on a dataset extrated using the *GeneMiner* application.

Process

Load the training and test datasets with pandas.

```
[ ]  import pandas as pd

     data_train = pd.read_csv("stomach_vs_skin_dataset_train.csv", index_col=0)

     data_test = pd.read_csv("stomach_vs_skin_dataset_test.csv", index_col=0)
```

Remove from the dataset attributes or samples that do not contain any information, i.e. all values are NAs.

```
[ ]  data_train = data_train.dropna(how='all')
     data_test = data_test.dropna(how='all')

     data_train = data_train.dropna(how='all', axis=1)
     data_test = data_test.dropna(how='all', axis=1)
```

Replace the remain NA values by the mean of the attribute.

```
[ ]  data_train = data_train.fillna(data_train.mean())
     data_test = data_test.fillna(data_test.mean())
```

Create pandas Dataframes to train and test the model.

```
[ ]  X_train = data_train.drop('CANCER', axis=1)
     y_train = data_train['CANCER']

     X_test = data_test.drop('CANCER', axis=1)
     y_test = data_test['CANCER']
```

Figure 5.15: Process use case. Using Jupyter transform the data from a formated file into a data structure.

5.1.7 Use Case 7: Modeling

Figure 5.16: Modeling use case. Use the data structure to create a model.

With the processed data from the previous step the researcher is able to create a model to predict the class of new samples. The framework supports the import of other ML frameworks that already implement ML algorithms that generate models For example *Scikit-learn* framework is a famous framework with a huge community and support, that easily allows the creation and training of models using popular ML algorithms. Figure 5.17 is a simple example of the creation of a model using a *Jupyter* notebook from our framework.

Modeling

Create and train a SVM model with the datasets

```
[ ]  from sklearn import svm

     svm_classifier = svm.SVC()

     svm_classifier.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/svm/base.py:196:
  "avoid this warning.", FutureWarning)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecate
  kernel='rbf', max_iter=-1, probability=False, random_state=Non
  shrinking=True, tol=0.001, verbose=False)
```

Figure 5.17: Training a SVM model on a dataset.

5.1.8 Use Case 8: Evaluate

Figure 5.18: Evaluate use case. Evaluate the model created.

After creating a model it urges the need to know how well it performs. There are several techniques used to test a model. Computing metrics that can finally tell us how good a model really is. For classification problems we have: accuracy, lost, precision, recall, f-measures, among others. Figure 5.19 is a simple example of an evaluation task completed in a *Jupyter* notebook.

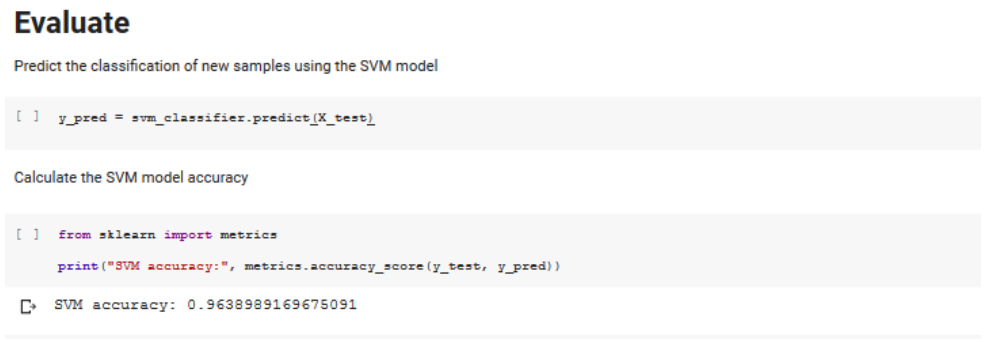


Figure 5.19: Evaluating a SVM model .

5.1.9 Use Case 9: Report

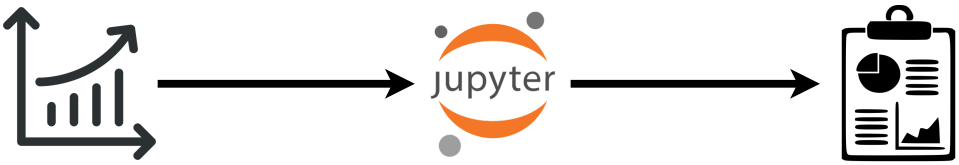
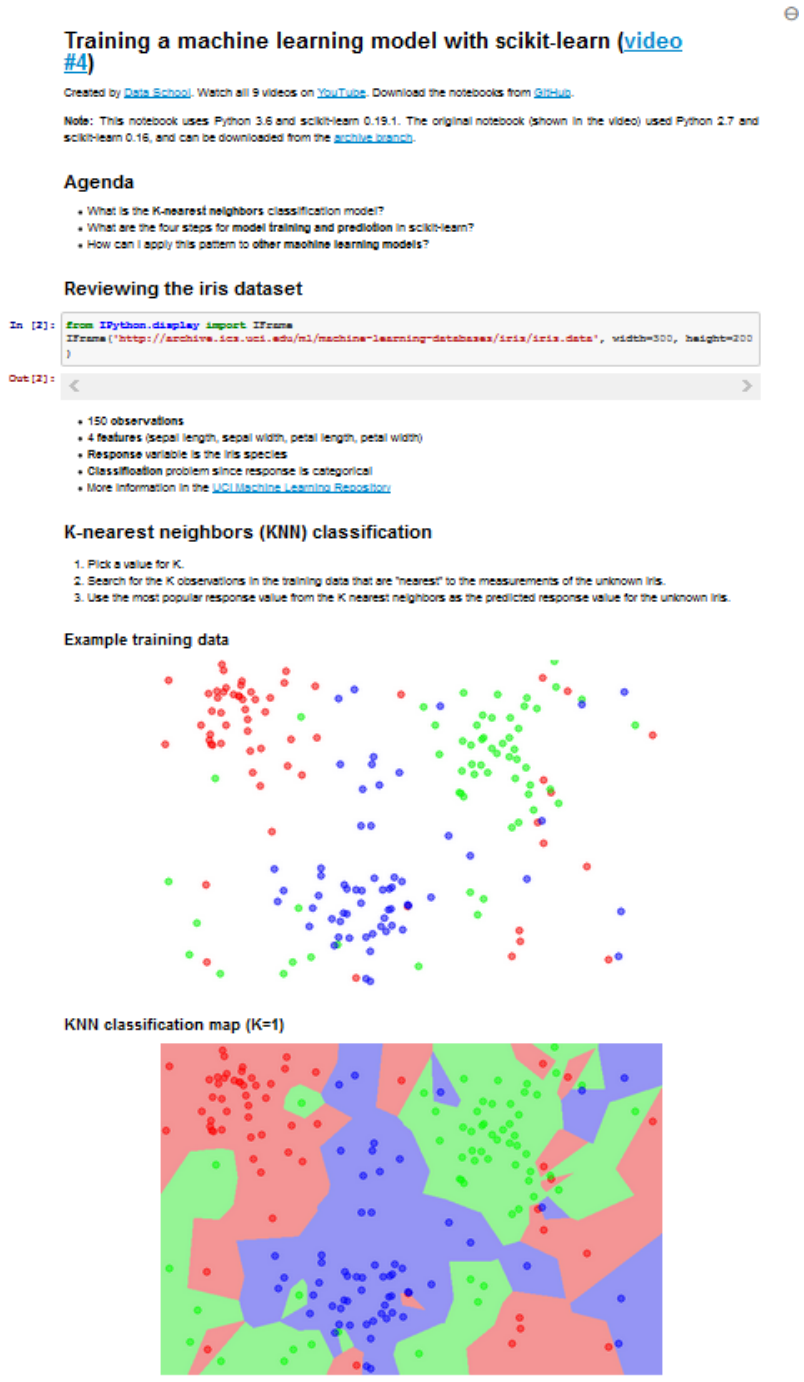


Figure 5.20: Report use case. Use Jupyter notebook as a report of the whole process.

At the end of each project it is very important to report all the work done and its results. The framework leverages the structure of *Jupyter* notebooks to showcase all the work done. The user is able to gradually build this report while working on the project and document each step of the process. A *Jupyter* notebook can display rich text, code, graphs, images, tables, formulas, among others. The user is responsible for documenting each step of the project and the report quality is directly connected to the documentation capability of the project owner. Figure 5.21 illustrates a well documented report using a *Jupyter* notebook.



Training a machine learning model with scikit-learn (video #4)

Created by Data School. Watch all 9 videos on YouTube. Download the notebooks from GitHub.

Note: This notebook uses Python 3.6 and scikit-learn 0.19.1. The original notebook (shown in the video) used Python 2.7 and scikit-learn 0.16, and can be downloaded from the archive branch.

Agenda

- What is the **K-nearest neighbors** classification model?
- What are the four steps for **model training and prediction** in scikit-learn?
- How can I apply this pattern to **other machine learning models**?

Reviewing the iris dataset

```
In [2]: from IPython.display import IFrame
        IFrame('http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', width=300, height=200
        )
Out[2]:
```

- 150 **observations**
- 4 **features** (sepal length, sepal width, petal length, petal width)
- **Response** variable is the iris species
- **Classification** problem since response is categorical
- More information in the UCI Machine Learning Repository

K-nearest neighbors (KNN) classification

1. Pick a value for K.
2. Search for the K observations in the training data that are "nearest" to the measurements of the unknown iris.
3. Use the most popular response value from the K nearest neighbors as the predicted response value for the unknown iris.

Example training data

KNN classification map (K=1)

Figure 5.21: Training a machine learning model with scikit-learn.[1]

[1] https://github.com/justmarkham/scikit-learn-videos/blob/master/04_model_training.ipynb

5.2 User Study

Although the use cases described in Section 5.1 evaluate the framework regarding its feature completion, it does not evaluate the results it can produce. To do such evaluation, this user study was performed to demonstrate the potential and convenience of using a tailored made framework to tackle a particular problem.

5.2.1 User Study: Problem Definition

The study objective is to use the framework to solve a particular problem. The goal is to build models that can distinguish between samples from two separate clinical trials: stomach and skin cancer. In each clinical trial three genomic experiments were conducted: Copy Number Analysis, Gene Expression and Human Methylation. Table 5.1 presents an overview of the data used in this study.

Table 5.1: Overview of the problem.

		Clinical Trial	
Experiments		Stomach Cancer	Skin Cancer
Copy Number Analysis	Samples	441	367
	Features	24777	24777
Gene Expression	Samples	415	472
	Features	20441	20441
Human Methylation	Samples	397	473
	Features	16765	16049

Four evaluation metrics are calculated from the output of the models trained with this dataset:

1. **Accuracy:** — $\frac{TruePositives+TrueNegatives}{TruePositives+TrueNegatives+FalsePositives+FalseNegatives}$;
2. **Precision:** — $\frac{TruePositives}{TruePositives+FalsePositives}$;
3. **Recall:** — $\frac{TruePositives}{TruePositives+FalseNegatives}$;
4. **F1 Score:** — $2\frac{precision*recall}{precision+recall}$;

5.2.2 User Study Case 1: Entire Dataset

This case study uses the export settings from Table 5.2 to retrieve a dataset with the entire space of features from both trials, using the *GeneMiner* application. That dataset will be used to train two models using SVMs and Random Forest algorithms. The evaluation results are presented in Table 5.3.

Table 5.2: Export settings used in *GeneMiner*.

Setting	**Value**
Exams to include	Copy Number Analysis, Gene Expression, Human Methylation
Validation Type	Hold out method, 70% training 30% test
Stacking	No
Feature Selection	No
File Type	*CSV*

Table 5.3: Model evaluation metrics.

		Model	
Metrics		SVM	Random Forest
Accuracy		0.9639	1.0
Stomach	Precision	1.0	1.0
	Recall	0.9254	1.0
	F1 Score	0.9612	1.0
Skin	Precision	0.9346	1.0
	Recall	1.0	1.0
	F1 Score	0.9662	1.0

5.2.3 User Study Case 2: Copy Number Analysis

This case study uses the export settings from Table 5.4 to retrieve a dataset with only the Copy Number Analysis features from both trials, using the *GeneMiner* application. That dataset will be used to train two models using SVMs and Random Forest algorithms. The evaluation results are presented in Table 5.5.

Table 5.4: Export settings used in *GeneMiner*.

Setting	**Value**
Exams to include	Copy Number Analysis
Validation Type	Hold out method, 70% training 30% test
Stacking	No
Feature Selection	No
File Type	*CSV*

Table 5.5: Model evaluation metrics.

		Model	
Metrics		SVM	Random Forest
Accuracy		0.9025	0.9170
Stomach	Precision	0.8963	0.9051
	Recall	0.9030	0.9253
	F1 Score	0.8996	0.9151
Skin	Precision	0.9085	0.9285
	Recall	0.9021	0.9091
	F1 Score	0.9052	0.9187

5.2.4 User Study Case 3: Gene Expression

This case study uses the export settings from Table 5.6 to retrieve a dataset with only the Gene Expression features from both trials, using the *GeneMiner* application. That dataset will be used to train two models using SVMs and Random Forest algorithms. The evaluation results are presented in Table 5.7.

Table 5.6: Export settings used in *GeneMiner*.

Setting	**Value**
Exams to include	Gene Expression
Validation Type	Hold out method, 70% training 30% test
Stacking	No
Feature Selection	No
File Type	*CSV*

Table 5.7: Model evaluation metrics.

		Model	
Metrics		SVM	Random Forest
Accuracy		0.8664	0.9964
Stomach	Precision	1.0	0.9926
	Recall	0.7238	1.0
	F1 Score	0.8398	0.9963
Skin	Precision	0.7944	1.0
	Recall	1.0	0.9930
	F1 Score	0.8854	0.9965

5.2.5 User Study Case 4: Human Methylation

This case study uses the export settings from Table 5.8 to retrieve a dataset with only the Human Methylation features from both trials, using the *GeneMiner* application. That dataset will be used to train two models using SVMs and Random Forest algorithms. The evaluation results are presented in Table 5.9.

Table 5.8: Export settings used in *GeneMiner*.

Setting	**Value**
Exams to include	Human Methylation
Validation Type	Hold out method, 70% training 30% test
Stacking	No
Feature Selection	No
File Type	*CSV*

Table 5.9: Model evaluation metrics.

		Model	
Metrics		SVM	Random Forest
Accuracy		1.0	1.0
Stomach	Precision	1.0	1.0
	Recall	1.0	1.0
	F1 Score	1.0	1.0
Skin	Precision	1.0	1.0
	Recall	1.0	1.0
	F1 Score	1.0	1.0

5.2.6 User Study Case 5: Feature Selection

This case study uses the export settings from Table 5.10 to retrieve a dataset with what supposedly are the best 500 features from both trials, using the *GeneMiner* application. That dataset will be used to train two models using SVMs and Random Forest algorithms. The evaluation results are presented in Table 5.11.

Table 5.10: Export settings used in *GeneMiner*.

Setting	**Value**
Exams to include	Copy Number Analysis, Gene Expression, Human Methylation
Validation Type	Hold out method, 70% training 30% test
Stacking	No
Feature Selection	Yes
File Type	*CSV*

Table 5.11: Model evaluation metrics.

Model	SVM	Random Forest
Accuracy	1.0	1.0
Stomach Precision	1.0	1.0
Skin Precision	1.0	1.0
Stomach Recall	1.0	1.0
Skin Recall	1.0	1.0
Stomach F1 Score	1.0	1.0
Skin F1 Score	1.0	1.0

5.2.7 User Study Case 6: Entire Dataset with K-fold CV

This case study uses the export settings from Table 5.12 to retrieve a dataset divided by 10 folds with the entire space of features from both trials, using the *GeneMiner* application. That dataset will be used to train two models using SVMs and Random Forest algorithms. The evaluation results are presented in Table 5.13.

Table 5.12: Export settings used in *GeneMiner*.

Setting	**Value**
Exams to include	Copy Number Analysis, Gene Expression, Human Methylation
Validation Type	K-fold Cross Validation, with a fold of 10
Stacking	No
Feature Selection	No
File Type	*CSV*

Table 5.13: Model evaluation metrics.

			Model	
Metrics			SVM	Random Forest
Mean	Accuracy		0.9423	0.9989
	Stomach	Precision	0.9921	0.9977
		Recall	0.8878	1.0
		F1 Score	0.9366	0.9988
	Skin	Precision	0.9052	1.0
		Recall	0.9936	0.9978
		F1 Score	0.9471	0.9989
Standard Deviation	Accuracy		0.0252	0.0032
	Stomach	Precision	0.0119	0.0066
		Recall	0.0446	0.0
		F1 Score	0.0287	0.0033
	Skin	Precision	0.0352	0.0
		Recall	0.0096	0.0063
		F1 Score	0.0224	0.0032

5.3 Conclusions

Section 5.1 demonstrated that all the use cases mentioned in Section 3.4 are completely satisfied. These features are the requirements of this project and represent the workflow of the platform from raw data to the final report. All use cases were fulfilled and represent a way to compare this project with existing similar tools in a pragmatic and objective manner.

The case study provides a more subjective approach, where a practical example is solved using this tool.

Globally, we are pleased with the evaluation of this project because it demonstrated that it is capable of the achieving the tasks it aimed to support and provide researchers with a framework for genomic based cancer studies using Machine Learning algorithms.

Chapter 6

Conclusion and Future Work

This chapter is a contemplation of the work done for this project. At first, we will enumerate the main difficulties that the project presented, refer the main contributions and conclusions achieved by this dissertation. Lastly, we will present possible future work on this project that could be done to improve and achieve a better framework.

6.1 Difficulties

Coming from a Computer Science background it is not easy to understand genomic data without an extensive and meticulous investigation on the field. Furthermore, frequently genomic data repositories do not have a standard format for the representation of data which represents a challenge in the preprocessing task if the user is new to the field.

The original data used in this project has a relatively small dimension, the amount of samples on each clinical trial was less than 500. Moreover, the samples that we had were all from sick patients. This presents a challenge for testing the framework in the sense that the results obtained are not that interesting, *i.e.* distinguish cancer samples. It would be better and far more interesting to predict whether a sample was cancerous or not.

6.2 Contributions

Our framework can solve all the questions presented in Chapter 3.5 under the same environment for a easy to use and workflow based tool. In here we present the answers to those questions based on our approach:

1. **Versatile user interface, enough to allow the user to preprocess raw data and to give it a structure:** — Using our framework the user has access the ability to create *Jupyter* notebooks that give that kind of versatility, providing the user with a built-in *Python* interpreter to solve these tasks;

2. **Minimum restrictions that one can apply to the data structure to be able to display the maximum variety of data:** — The strucuter defined in this project allow the use of a *MogoDB* component to have each clinical trial in its own *Mongo* database and each of the experiments be a collection of that database;

3. **Important information to display:** — It is generalized to some metrics specific to the context (*eg.* total and unique subjects or samples) and to statistics (*eg.* mean, median, maximum, minimum, distribution);

4. **Powerful enough user interface to allow the user to process, model, evaluate and export the users work:** — The framework allows the use of a *Jupyter* notebook that combines *Python* and *Markdown* to produce a report like document capable of process and model data;

5. **User interface from 1 and 4 combined:** — Our framework can manage multiple *Jupyter* notebooks, meaning that for all the computational tasks this project leverages the resources of *Jupyter* notebooks.

6.3 Conclusions

Our objective was to provide geneticist with a framework for genomic based cancer studies using Machine Learning algorithms. We successfully built a workflow based approach to tackle this issue with a customizable analysis pipeline and reviewed the contribution of this project in the extraction of knowledge from genomic data.

We can state that the objective of this project was completely fulfilled. However, this was a research work with proof of concept features and there is work that can be done to improve the entire pipeline.

6.4 Future Work

There is still a lot of work that can be done to improve our solution. As it is, the entire framework is an open system composed by three different components. Thus, the user needs to know how the pipeline exchanges data from one component to another in order to use it. This is a drawback on the convenience and can disrupt the overall experience. So, one important improvement should be to close the system convening the idea of a single application.

The *GeneMiner* application could be improved to allow a user to import data directly from a file or an existing genomic data repository. Futhermore, the visualization of data could use an improvement since it only displays basic information about trials, experiments and features. A higher number of settings could be provided for the user to extract a dataset.

On a side note, the ability to generate datasets for stacking models was not validated by our user study. Hence, this could be something to validate in the future.

References

[1] Cancer. https://www.who.int/en/news-room/fact-sheets/detail/cancer. Accessed: 2019-01-17.

[2] Cancer - signs and symptoms - nhs. https://www.nhs.uk/conditions/cancer/symptoms/. Accessed: 2019-01-17.

[3] Cross validation. https://www.cs.cmu.edu/~schneide/tut5/node42.html. Accessed: 2019-01-17.

[4] What is cancer? - national cancer institute. https://www.cancer.gov/about-cancer/understanding/what-is-cancer. Accessed: 2019-01-17.

[5] What is dna? - genetics home reference - nih. https://ghr.nlm.nih.gov/primer/basics/dna. Accessed: 2019-01-17.

[6] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[7] Iñigo Barandiaran. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell*, 20(8), 1998.

[8] Ardeshir Bayat. Science, medicine, and the future: Bioinformatics. *BMJ: British Medical Journal*, 324(7344):1018, 2002.

[9] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[10] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[11] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[12] Gavin C Cawley and Nicola LC Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(Jul):2079–2107, 2010.

[13] Suzanne Clancy. Rna functions. *Nat Educ*, 1(1):102, 2008.

[14] Geoffrey M Cooper, Robert E Hausman, and Robert E Hausman. *The cell: a molecular approach*, volume 2. ASM press Washington, DC, 2000.

[15] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[16] Vasant Dhar. Data science and prediction. *Communications of the ACM*, 56(12):64–73, 2013.

[17] Sean B Eom. Decision support systems.[in]: International encyclopedia of business and management, warner m. *International Thomson Business Publishing Co., London*, 2001.

[18] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[19] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[20] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[21] Amit X Garg, Neill KJ Adhikari, Heather McDonald, M Patricia Rosas-Arellano, PJ Devereaux, Joseph Beyene, Justina Sam, and R Brian Haynes. Effects of computerized clinical decision support systems on practitioner performance and patient outcomes: a systematic review. *Jama*, 293(10):1223–1238, 2005.

[22] Seymour Geisser. *Predictive inference*. Routledge, 2017.

[23] Corrado Gini. Concentration and dependency ratios. *Rivista di politica economica*, 87:769–792, 1997.

[24] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[25] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[26] Andreas Holzinger. Weakly structured data in health-informatics: The challenge for human-computer interaction. In *Proceedings of INTERACT 2011 Workshop: Promoting and supporting healthy living by design*. ., 2011.

[27] Earl B Hunt, Janet Marin, and Philip J Stone. Experiments in induction. 1966.

[28] Peter GW Keen. Decisión support systems; an organizational perspective. Technical report, 1978.

[29] Peter GW Keen. Decision support systems: a research perspective. In *Decision Support Systems: Issues and Challenges: Proceedings of an International Task Force Meeting*, pages 23–44, 1980.

[30] James M Keller, Michael R Gray, and James A Givens. A fuzzy k-nearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, (4):580–585, 1985.

[31] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[32] Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, 23(1):89–109, 2001.

[33] Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003.

[34] Joseph Carl Robnett Licklider. Man-computer symbiosis. *IRE transactions on human factors in electronics*, (1):4–11, 1960.

[35] Geoffrey McLachlan. *Discriminant analysis and statistical pattern recognition*, volume 544. John Wiley & Sons, 2004.

[36] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[37] Nils J Nilsson. Learning machines. 1965.

[38] Mourad Ouzzani, Paolo Papotti, and Erhard Rahm. Introduction to the special issue on data quality. *Information Systems*, 2013.

[39] Jonathan Pevsner. *Bioinformatics and functional genomics*. John Wiley & Sons, 2015.

[40] Robi Polikar. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3):21–45, 2006.

[41] Daniel J Power. A brief history of decision support systems.

[42] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[43] Amy Ralston and Kenna Shaw. Gene expression regulates cell differentiation. *Nat Educ*, 1(1):127–31, 2008.

[44] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[45] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[46] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[47] Henk G Sol, A Th Cees, Pieter F de Vries Robbé, et al. *Expert systems and artificial intelligence in decision support systems: proceedings of the Second Mini Euroconference, Lunteren, The Netherlands, 17–20 November 1985*. Springer Science & Business Media, 2013.

[48] Peter Sollich and Anders Krogh. Learning with ensembles: How overfitting can be useful. In *Advances in neural information processing systems*, pages 190–196, 1996.

[49] Ralph H Sprague Jr. A framework for the development of decision support systems. *MIS quarterly*, pages 1–26, 1980.

[50] Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

[51] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[52] Tan Yigitcanlar. *Knowledge-Based Urban Development: Planning and Applications in the Information Era: Planning and Applications in the Information Era*. IGI Global, 2008.